***PSTYRO-FPGASP3***

***SPARTAN3 Starter Kit***

***Hardware & Software User Manual***

## *Contents*

# 1. *Introduction*

Thank you for purchasing the Xilinx Spartan-3 Evaluation Kit. You will find it useful in developing your Spartan-3 FPGA application.

FPGASP3 Kit (STK) is an exclusive general-purpose kit for the SPARTAN family. The intention of the design is to endorse the engineers and scholars to exercise and explore the capabilities of FPGA architectures with many interfacing modules on board point LEDs, Slide switches, UART, VGA, and PS/2 with ease to create a stand-alone versatile test platform.

---

## *1.1 – Packages*

- Spartan3 Starter Kit (XC3S200)
- Serial Port Cable
- JTAG Programming Cable
- Printed User Manual
- CD contains
    - Software (Programmers, ISE)
    - Example Programs
    - User Manual

---

## *1.2 - Technical or Customer Support*

E-mail questions to : support@pantechsolutions.net

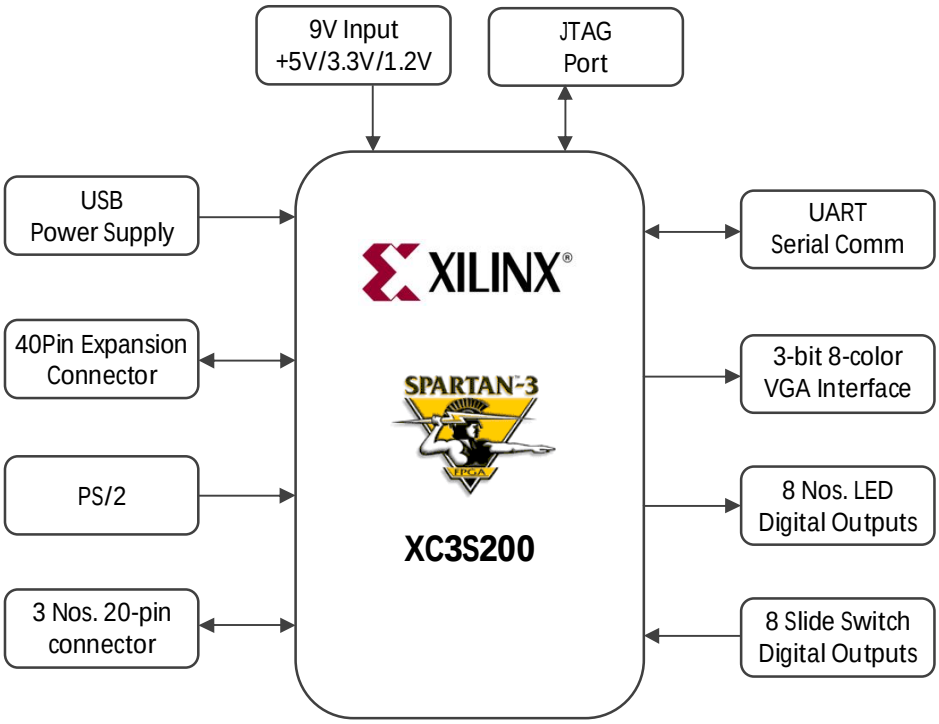
Send questions by mail to

Pantech Solutions Pvt Ltd.,
151/34, Mambalam High Road,
Sri Ranga Building, T. Nagar,
Chennai - 600 017
Tamilnadu, India
Phone : +91-44-4260 6470
Fax : +91-44-4260 6350
Website : www.pantechsolutions.net

## 2. *Key Components and Features*

- 200,000-gate Xilinx Spartan 3 FPGA in a 144-TQFP (XC3S200-4TQG144C)
  - 4,320 logic cell equivalents
  - Twelve 18K-bit block RAMs (216K bits)
  - Twelve 18x18 hardware multipliers
  - Four Digital Clock Managers (DCMs)
  - Up to 97 user-defined I/O signals
- 3-bit, 8-color VGA display port.
- RS-232 Serial Port.
  - DB9 9-pin male connector with RS-232 transceiver/level translator
  - Uses straight-through serial cable to connect PC or workstation serial port
- PS/2-connector( for mouse/keyboard interface) port
- 8 Nos. Slide Switches for digital inputs
- 8 nos. of Point LEDs for Digital outputs
- 50 MHz crystal oscillator clock source
- 3 Nos. 20-pin I/o connector for interface external peripherals modules
- 40-pin Expansion connector for interface additional i/o modules
- JTAG port for download user program through cable
- 9V AC/DC power input through adapter
- Power-on indicator LED
- On-board 5V, 3.3V, 2.5V, and 1.2V regulators.

## *2.1 - General Block Diagram*

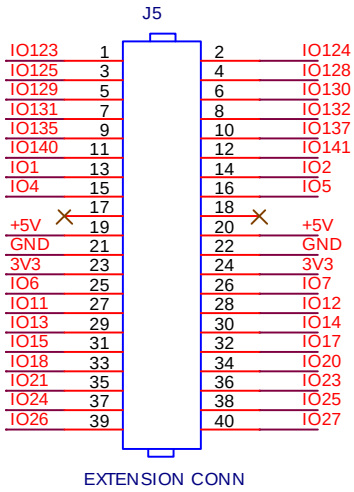


## 3. *Jumper Details*

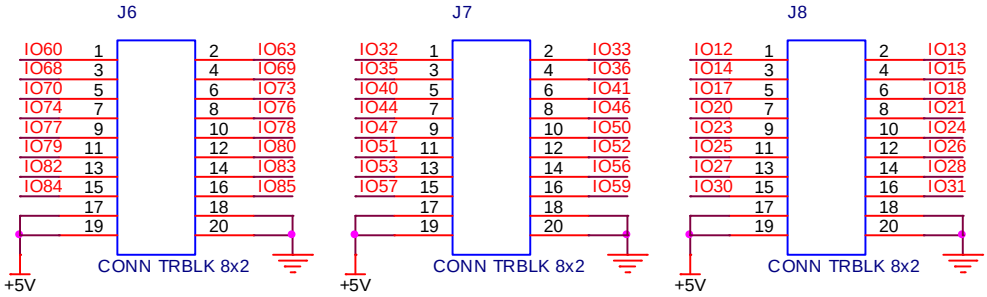
| | Jumper and Main Switch Selection Details | | |
|---|---|---|---|
| **JTAG/ PROM** | **J3** |  | PROM Execution<br>JTAG Executions |

## 4. *Connector Details*

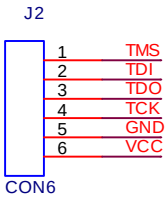
### 40-Pin Expansion Connector




### 20pin – Box Connector



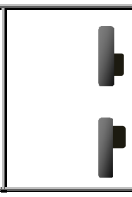

### JTAG Connector



## 5. *Power Supply*

The external power can be AC or DC, with a voltage between (9V, 1A output) at 230V AC input. The SPARTAN3 board produces +5V using an LM7805 voltage regulator, which provides supply to the peripherals.

USB socket meant for power supply and USB communication, user can select either USB or Ext power supply through **SW1**. Separate **On/Off** Switch (SW1) for controlling power to the board.

| | | |
|---|---|---|
| **ON/OFF**<br>SW1 |  | Power +5V ***ON*** **-**External through ***Adaptor***<br><br>Power +5V ***ON*** **-** Internal through ***USB*** |

There are multiple voltages supplied on the Spartan-3 Evaluation Kit, 3.3V, 2.5V and 1.2V regulators. Similarly, the 3.3V regulator feeds all the VCCO voltage supply inputs to the FPGA's I/O banks and powers most of the components on the board. The 2.5V regulator supplies power to the FPGA's VCCAUX supply inputs. The VCCAUX voltage input supplies power to Digital Clock Managers (DCMs) within the FPGA and supplies some of the I/O structures.

In specific, all of the FPGA's dedicated configuration pins, such as DONE, PROG_B, CCLK, and the FPGA's JTAG pins, are powered by VCCAUX. The FPGA configuration interface on the board is powered by 3.3V. Consequently, the 2.5V supply has a current shunt resistor to prevent reverse current. Finally, a 1.2V regulator supplies power to the FPGA's VCCINT voltage inputs, which power the FPGA's core logic. The board uses four discrete regulators to generate the necessary voltages.

## 6. *On-board Peripherals*

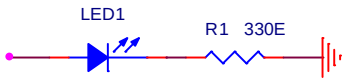
The Evaluation Kit comes with many interfacing options

- 8-Nos. of Point LED's (Digital Outputs)
- 8-Nos. of Digital Inputs (Slide Switches)
- UART for serial port communication through PC
- JTAG Programmer
- Clock Source
- 3-bit, 8 Color VGA Interface
- PS/2 Keyboard interface

## *6.1 - Light Emitting Diodes*

- Light Emitting Diodes (LEDs) are the most commonly used components, usually for displaying pin's digital states.

- The FPGASP3 KIT has 8 nos., of Point LEDs, connected with port pins (details tabulated below); the cathode of each LED connects to ground via a 220Ω resistor. To light an individual LED, drive the associated FPGA control signal to **High.**

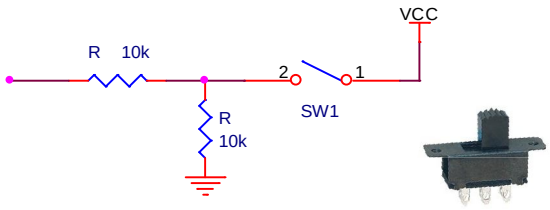
| | Point LEDs | SPARTAN3 FPGA Lines | LED Selection |
|---|---|---|---|
| DIGITAL OUTPUTS | LED-D11 | P97 |  <br>Make Pin High – LED ON<br>Make Pin Low – LED OFF |
| | LED-D10 | P98 | |
| | LED-D9 | P99 | |
| | LED-D8 | P100 | |
| | LED-D7 | P102 | |
| | LED-D6 | P103 | |
| | LED-D5 | P104 | |
| | LED-D4 | P105 | |

## *6.2 – Digital Inputs*

- This is another simple interface, of 8-Nos. of slide switch, mainly used to give an input to the port lines, and for some control applications also.

- The FPGASP3 KIT, slide switches (SW3-SW10) directly connected with FPGA I/O lines (details tabulated below), user can give logical inputs **high** through slide switches.

The switches are connected to +3.3V, in order to detect a switch state, by default lines are pull-downed through resistors. The switches typically exhibit about 2 ms of mechanical bounce and there is no active de-bouncing circuitry, although such circuitry could easily be added to the FPGA design programmed on the board. A 10KΩ series resistor provides nominal input protection.

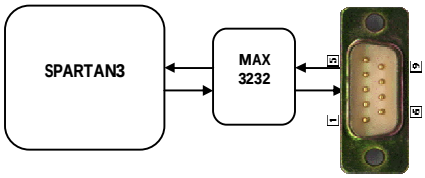
| | Slide Switches | SPARTAN3 FPGA Lines | Slide Switch Logic |
|---|---|---|---|
| DIGITAL INTPUTS | SW3 | P86 |  Make Switch Close – High<br>Make Switch Open – Low |
| | SW4 | P87 | |
| | SW5 | P89 | |
| | SW6 | P90 | |
| | SW7 | P92 | |
| | SW8 | P93 | |
| | SW9 | P95 | |
| | SW10 | P96 | |

## *6.3 - RS-232 Communication(USART)*

USART stands for Universal Synchronous Asynchronous Receiver Transmitter. FPGASP3 Kit provides an RS232 port that can be driven by the Spartan-3 FPGA. A subset of the RS232 signals is used on the Spartan 3 kit to implement this interface (RxD and TxD signals).

- RS-232 communication enables point-to-point data transfer. It is commonly used in data acquisition applications, for the transfer of data between the microcontroller/FPGA and a PC.

- The voltage levels of a FPGA and PC are not directly compatible with those of RS-232, a level transition buffer such as MAX3232 be used.

| | UART DB-9 Connector | SPARTAN3 FPGA Lines | Serial Port Section |
|---|---|---|---|
| UART | TXD | P122 |  |
| | RXD | P127 | |

## *6.4 – JTAG Programmer*

The FPGASP3 Kit includes a JTAG programming and debugging chain. Pantech JTAG3 low-cost parallel to JTAG cable is included as part of the kit and connects to the JTAG header. DB-25 parallel port connector to 6 pin female header connector. The JTAG cable connect directly to the parallel port of a PC and to a standard 6 pin JTAG programming header in the kit, can program a devices that have a JTAG voltage of 1.8V or greater.

| | 6-Pin Header | JTAG Signals | SPARTAN3 FPGA Lines | JTAG Cable – III |
|---|---|---|---|---|
| JTAG Programmer | 1 | TMS | P111 |  |
| | 2 | TDI | P144 | |
| | 3 | TDO | P109 | |
| | 4 | TCK | P110 | |
| | 5 | GND | | |
| | 6 | VCC | | |

The Pantech low-cost parallel port to JTAG cable fits directly over the header stake pins, as shown in above figure. When properly fitted, the cable is perpendicular to the board. Make sure that the signals at the end of the JTAG cable align with the labels listed on the board. The other end of the Pantech cable connects to the PC's parallel port. The Pantech cable is directly compatible with the Xilinx iMPACT software.

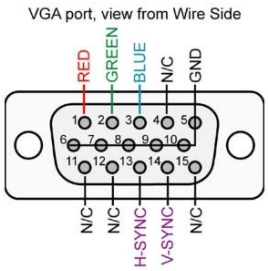
## *6.5 – Clock Source*

The FPGASP3 Kit has a dedicated 50 MHz series clock oscillator source and an optional socket for another clock oscillator source.

| | U10 | Signal | SPARTAN3 FPGA Lines | Crystal Oscillator |
|---|---|---|---|---|
| Oscillator | 50MHz | Clock | P55 |  |

## *6.6 - VGA Interface*

The FPGASP3 Kit includes a VGA display port through DB15 connector, Connect this port directly to most PC monitors or flat-panel LCD displays using a standard monitor cable As shown in table, the Spartan-3 FPGA controls five VGA signals: RED (R) its 1ST pin in connector, GREEN (G) its 2nd pin, BLUE (B) its 3rd pin, Horizontal Sync (HS) 13th pin, and Vertical Sync (VS) its 14th pin, all available on the VGA connector.

| DB-15 Connector | VGA Signals | SPARTAN3 FPGA Lines | |
|---|---|---|---|
|  | Vertical Sync(VS) | P12 |   |
| | Horizontal Sync(HS) | P13 | |
| | Blue | P16 | |
| | Green | P18 | |
| | Red | P19 | |

Each color line has a series resistor to provide 3-bit color, with one bit each for Red, Green, and Blue. The series resistor uses the 75 ohm VGA cable termination to ensure that the color signals remain in the VGA-specified 0V to 0.7V range. The HS and VS signals are TTL level. Drive the R, G, and B signals High or Low to generate the eight possible colors shown in below table.

| RED | GREEN | BLUE | RESULTING COLOUR |
|---|---|---|---|
| 0 | 0 | 0 | BLACK |
| 0 | 0 | 1 | BLUE |
| 0 | 1 | 0 | GREEN |
| 0 | 1 | 1 | CYAN |
| 1 | 0 | 0 | RED |
| 1 | 0 | 1 | MAGNETA |
| 1 | 1 | 0 | YELLOW |
| 1 | 1 | 1 | WHITE |

VGA signal timing is specified, published, copyrighted, and sold by the Video Electronics Standards Association (VESA). The following VGA system and timing information is provided as

an example of how the FPGA might drive VGA monitor in 640 by 480 modes. For more precise information or for information on higher VGA frequencies, refer to documents available on the VESA website or other electronics Websites: Video Electronics Standards Association

http://www.vesa.org

VGA Timing Information
http://www.epanorama.net/documents/pc/vga_timing.html

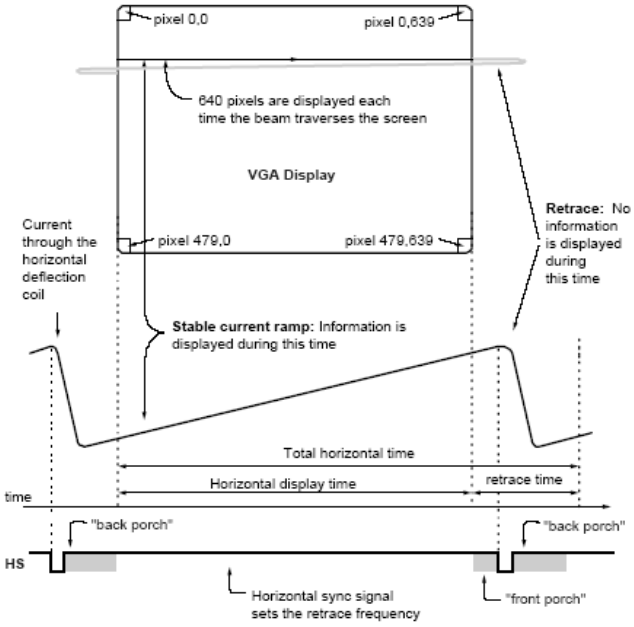
Signal Timing for a 60Hz, 640x480 VGA Display

CRT-based VGA displays use amplitude-modulated, moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel by- pixel basis.

Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays. Consequently, the following discussion pertains to both CRTs and LCD displays. Within a CRT display, current waveforms pass through the coils to produce magnetic fields that deflect electron beams to transverse the display surface in a "raster" pattern, horizontally from left to right and vertically from top to bottom.

As shown in below figure, information is only displayed when the beam is moving in the "forward" direction—left to right and top to bottom—and not during the time the beam returns back to the left or top edge of the display. Much of the potential display time is therefore lost in "blanking" periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass.

The size of the beams, the frequency at which the beam traces across the display, and the frequency at which the electron beam is modulated determine the display resolution. Modern VGA displays support multiple display resolutions, and the VGA controller indicates the resolution by producing timing signals to control the raster patterns. The controller produces TTL-level synchronizing pulses that set the frequency at which current flows through the deflection coils, and it ensures that pixel or video data is applied to the electron guns at the




correct time. Video data typically comes from a video refresh memory with one or more bytes assigned to each pixel location.

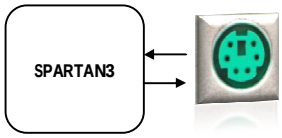
The Spartan-3 Evaluation Kit uses three bits per pixel, producing one of the eight possible colors shown in above table. The controller indexes into the video data buffer as the beams move across the display. The controller then retrieves and applies video data to the display at precisely the time the electron beam is moving across a given pixel.

The VGA controller generates the HS (horizontal sync) and VS (vertical sync) timings signals and coordinates the delivery of video data on each pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the “refresh” frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display’s phosphor and electron beam intensity, with practical refresh frequencies in the 60 Hz to 120 Hz range.

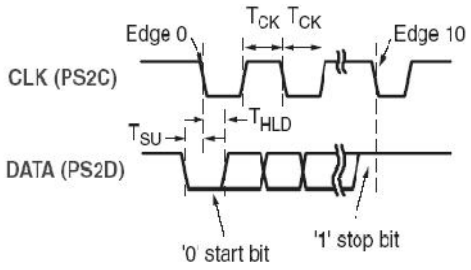
The number of horizontal lines displayed at a given refresh frequency defines the horizontal “retrace” frequency.

## *6.7 - PS/2 Interface*

The FPGASP3 Kit includes a PS/2 port and the standard 6-pin mini-DIN connector, labeled U11 on the board. User can connect PS/2 Devices like keyboard, mouse to the FPGASP3 KIT. PS/2's DATA (P8) and CLK (P10) lines connected to SPARTAN3 FPGA I/O Lines.

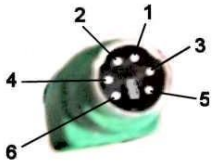
| 6PIN MINI Connector | PS/2 | SPARTAN3 FPGA Lines | PS/2 PORT SELECT |
|---|---|---|---|
| U11 PS/2 | DATA | P8 |  |
| | CLK | P10 | |






**PS/2 keyboard connector (MINI-DIN6)**

| Connector Pin # | Purpose |
|---|---|
| Pin 1 | KBDAT (data) |
| Pin 2 | not used |
| Pin 3 | GND |
| Pin 4 | VCC (+5V) |
| Pin 5 | KBDCLK (clock) |
| Pin 6 | not used |




ıouse connector pinout is identical to PS/2 keyboard.

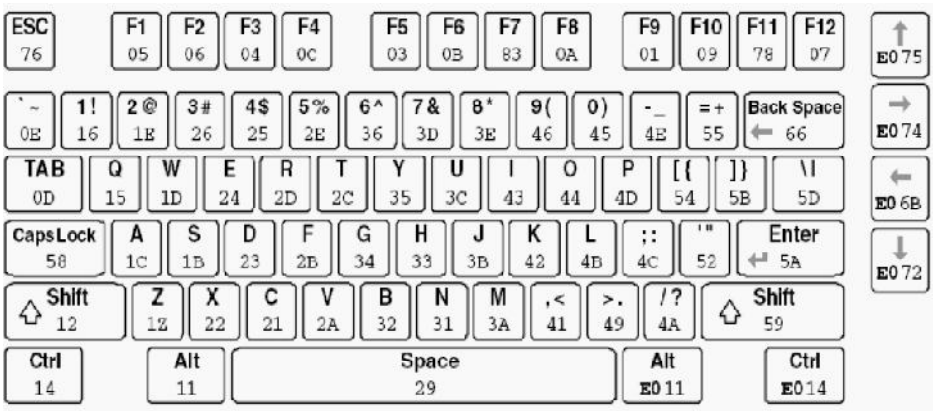
Both a PC mouse and keyboard use the two-wire PS/2 serial bus to communicate with a host device, the Spartan-3 FPGA in this case. The PS/2 bus includes both clock and data. Both a mouse and keyboard drive the bus with identical signal timings and both use 11-bit words that include a start, stop and odd parity bit. However, the data packets are organized differently for a mouse and keyboard. Furthermore, the keyboard interface allows bidirectional data transfers so the host device can illuminate state LEDs on the Keyboard.

The PS/2 bus timing appears as shown in above figure. The clock and data signals are only driven when data transfers occur; otherwise they are held in the idle state at logic High. The timing defines signal requirements for mouse-to-host communications and bidirectional keyboard communications. The attached keyboard or mouse writes a bit on the data line when the clock signal is High, and the host reads the data line when the clock signal is Low.

***Keyboard***

The keyboard uses open-collector drivers so that either the keyboard or the host can drive the two-wire bus. If the host never sends data to the keyboard, then the host can use simple input pins. A ps/2-style keyboard uses scan codes to communicate key press data nearly all keyboards in use today are ps/2 style. Each key has a single, unique scan code that is sent whenever the corresponding key is pressed.

The scan codes for most keys appear in below figure. If the key is pressed and held, the keyboard repeatedly sends the scan code every 100 ms or so. When a key is released, the keyboard sends an "f0" key-up code, followed by the scan code of the released key. the keyboard sends the same scan code, regardless if a key has different shift and non-shift characters and regardless whether the shift key is pressed or not. The host determines which character is intended. Some keys, called extended keys, send an "e0" ahead of the scan code and furthermore, they might send more than one scan code. When an extended key is released, an "e0 f0" key-up code is sent, followed by the scan code.



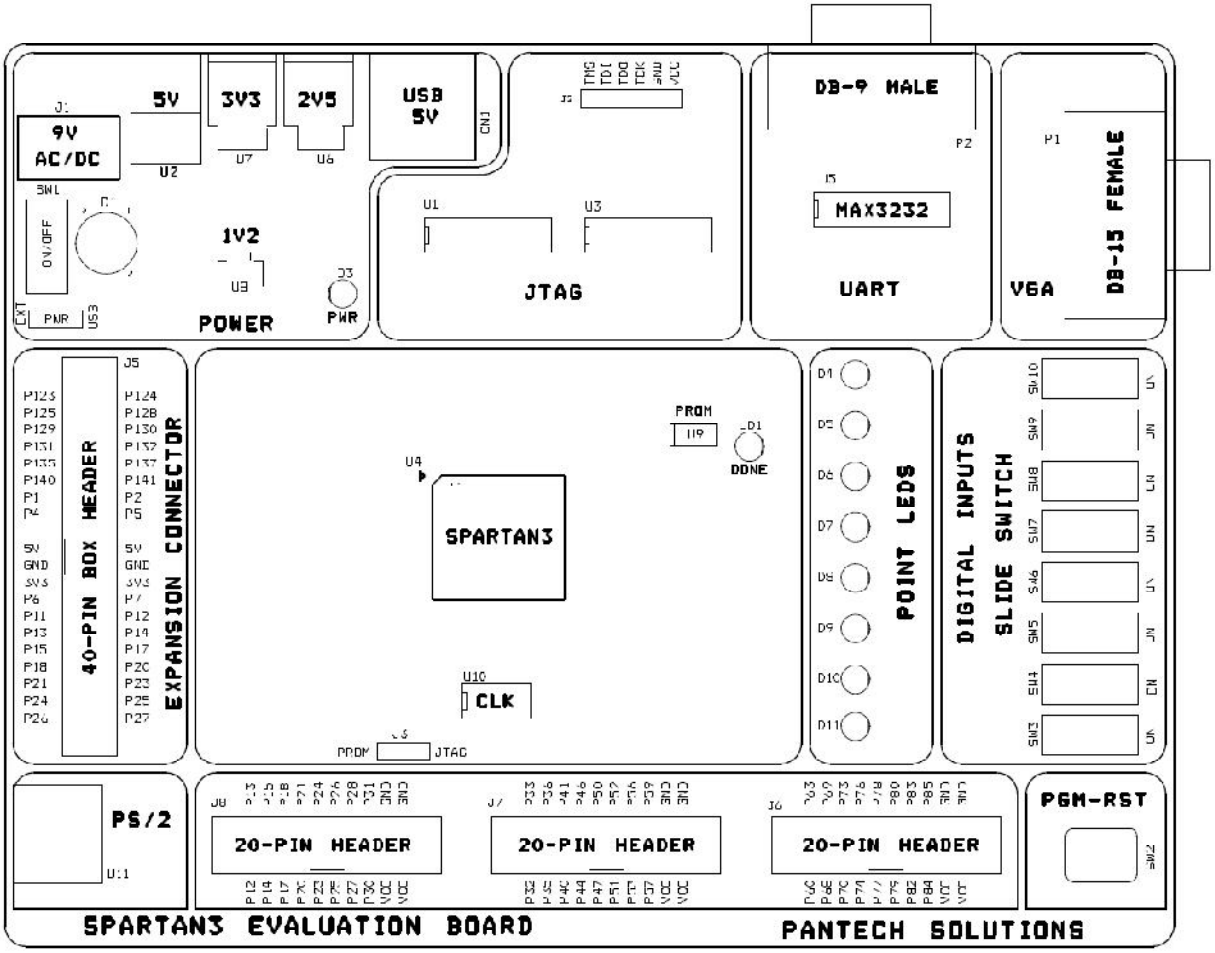

The host can also send commands and data to the keyboard. Below figure provides a short list of some often-used

***Commands***

| Command | Description |
|---|---|
| ED | Turn on/off Num Lock, Caps Lock, and Scroll Lock LEDs |
| EE | Echo. Upon receiving an echo command, the keyboard replies with the same scan code "EE". |
| F3 | Set scan code repeat rate. The keyboard acknowledges receipt of an "F3" by returning an "FA", after which the host sends a second byte to set the repeat rate. |
| FE | Resend. Upon receiving a resend command, the keyboard resends the last scan code sent |
| FF | Reset. Resets the keyboard |

The keyboard sends commands or data to the host only when both the data and clock lines are High, the Idle state, Because the host is the bus master, the keyboard checks whether the host is sending data before driving the bus. The clock line can be used as a clear to send signal. If the host pulls the clock line Low, the keyboard must not send any data until the clock is released. The keyboard sends data to the host in 11-bit words that contain a '0' start bit, followed by eight bits of scan code (LSB first), followed by an odd parity bit and terminated with a '1' stop bit.

## 7. *Board Layout*



# *VLSI LAB Examples*

***(L-Scheme)***

## Contents

### *8. VLSI Lab Experiments*

## SIMULATION OF VHDL CODE FOR COMBINATIONAL CIRCUIT (Sum of Product)

# 1A

**Description**

Optimize a 4 variable combinational function (SOP or POS), describe it in VHDL code and Simulate it.
Example: F= (0, 5, 8, 9, 12) in sop

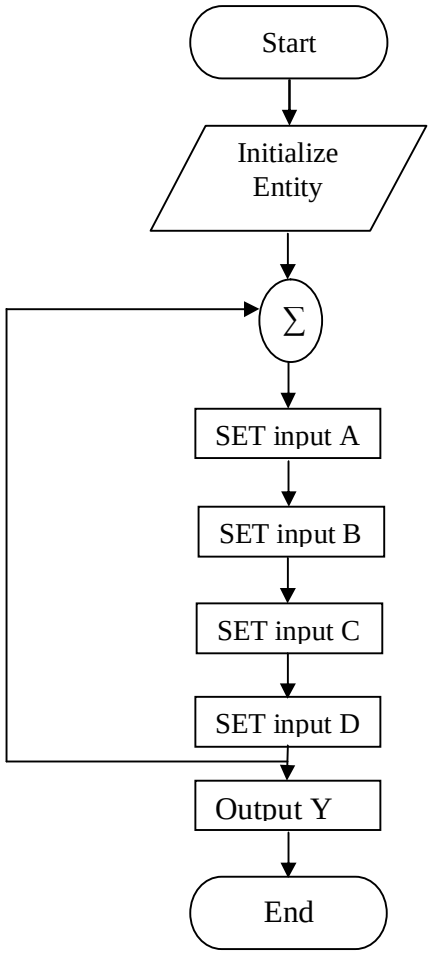
**Truth Table for Sum of Product Simplification**

| A | B | C | D | Y | Sum of Product |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | A'B'C'D' |
| 0 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 0 | 0 | |
| 0 | 0 | 1 | 1 | 0 | |
| 0 | 1 | 0 | 0 | 0 | |
| 0 | 1 | 0 | 1 | 1 | A'BC'D |
| 0 | 1 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 1 | AB'C'D' |
| 1 | 0 | 0 | 1 | 1 | AB'C'D |
| 1 | 0 | 1 | 0 | 0 | |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 0 | 1 | ABC'D' |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | 0 | |

**Boolean Expression:**

Y = A'B'C'D'+A'BC'D+AB'C'D'+AB'C'D+ABC'D'
Y = B'C'D'(A'+A) +C'(A'BD+AB'D+ABD')
Y = B'C'D'+A'BC'D+AB'C'D+ABC'D'

**Flow Chart**

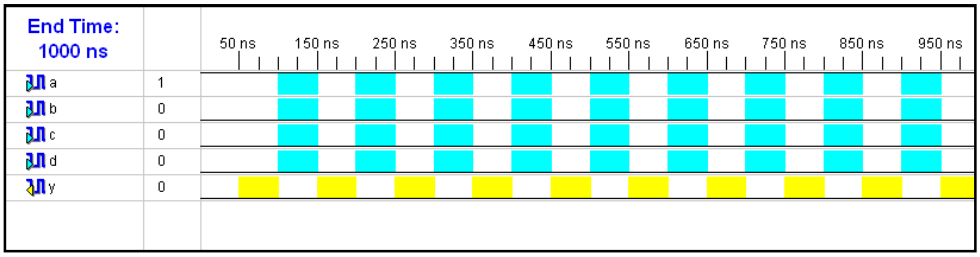

Start
Initialize Entity
Σ
SET input A
SET input B
SET input C
SET input D
Output Y
End

**Code Listing**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sop is
port(a,b,c,d : in std_logic;
     y : out std_logic
);
 end sop;

architecture data of sop is

begin

 y <= (( not b and not c and not d) or (not a and b and not c and
d) or (a and not b and not c and  d) or (a and  b and not c and not
d) );

end data;
```

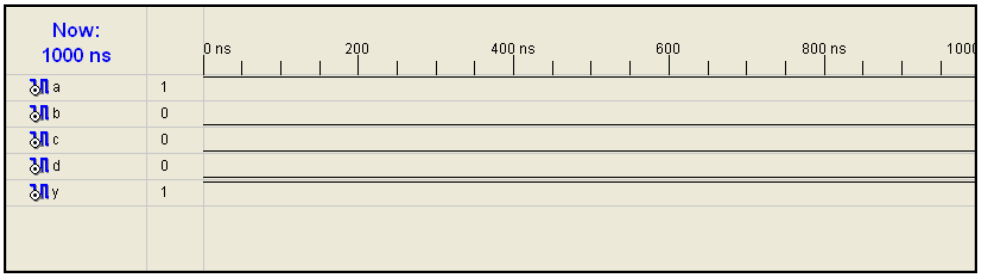
## Input Waveform

We give input a=high in waveform window




## Output waveform

Finally we get output in simulation window(y=high according to that SOP Equation)



## SIMULATION OF VHDL CODE FOR COMBINATIONAL CIRCUIT (Product of Sum)

# 1.B

**Description**

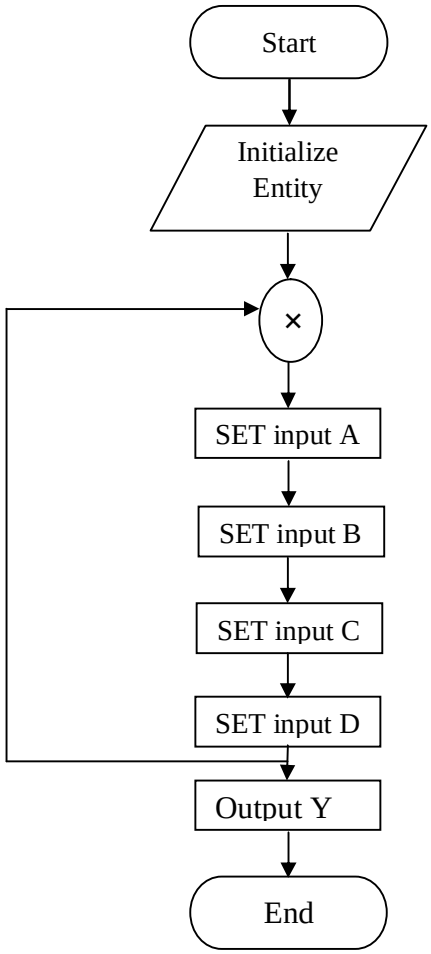
Optimize a 4 variable combinational function (POS), describe it in VHDL code and Simulate it.
Example: F= (0, 5, 8, 9, 12) in POS.

**Truth Table for Product of Sum Simplification**

| A | B | C | D | Y | Product of Sum |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | |
| 0 | 0 | 0 | 1 | 0 | A'+B'+C'+D |
| 0 | 0 | 1 | 0 | 0 | A'+B'+C+D' |
| 0 | 0 | 1 | 1 | 0 | A'+B'+C+D |
| 0 | 1 | 0 | 0 | 0 | A'+B+C'+D' |
| 0 | 1 | 0 | 1 | 1 | |
| 0 | 1 | 1 | 0 | 0 | A'+B+C+D' |
| 0 | 1 | 1 | 1 | 0 | A'+B+C+D |
| 1 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 1 | |
| 1 | 0 | 1 | 0 | 0 | A+B'+C+D' |
| 1 | 0 | 1 | 1 | 0 | A+B'+C+D |
| 1 | 1 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 1 | 0 | A+B+C'+D |
| 1 | 1 | 1 | 0 | 0 | A+B+C+D' |
| 1 | 1 | 1 | 1 | 0 | A+B+C+D |

Y = (A'+B'+C'+D ) (A'+B'+C+D') (A'+B'+C+D) (A'+B+C'+D') (A'+B+C+D') (A'+B+C+D) (A+B'+C+D') (A+B'+C+D) (A+B+C'+D) (A+B+C+D') (A+B+C+D)

**Flow Chart**

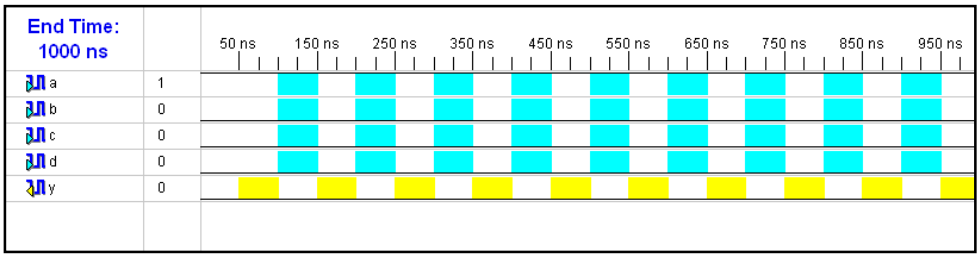

Start
Initialize Entity
×
SET input A
SET input B
SET input C
SET input D
Output Y
End

**Code Listing**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pos is
port(a,b,c,d : in std_logic;
     y : out std_logic
);
 end pos;

architecture data of pos is

begin

 y <= ((not a or not b or not c or  d ) and(not a or not b or  c or
not  d) and (not a or not b or  c or  d) and (not a or   b or not c
or not d)and (not a or b or  c or not d)and(not a or b or c or d)
and (a or not b or c not d) and (a or not b or c or d)and(a or b or
not c or d  ) and (a or b or c or not d) and (a or b  or c or d)) ;

end data;
```

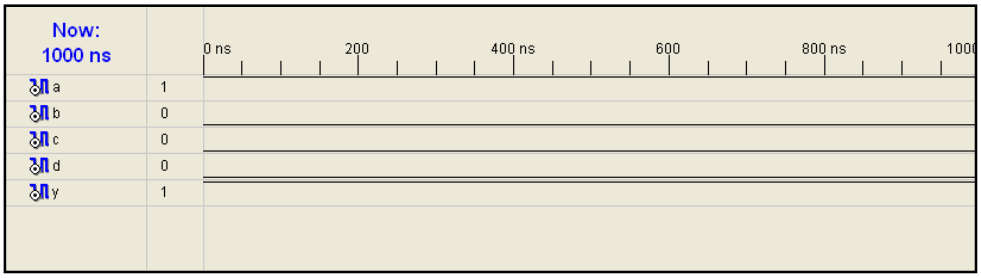
## Input Waveform

We give input a=high in waveform window




## Output waveform

Finally we get output in simulation window(y=high according to that POS Equation)



# SIMULATION OF VHDL CODE FOR ARITHMETIC CIRCUITS

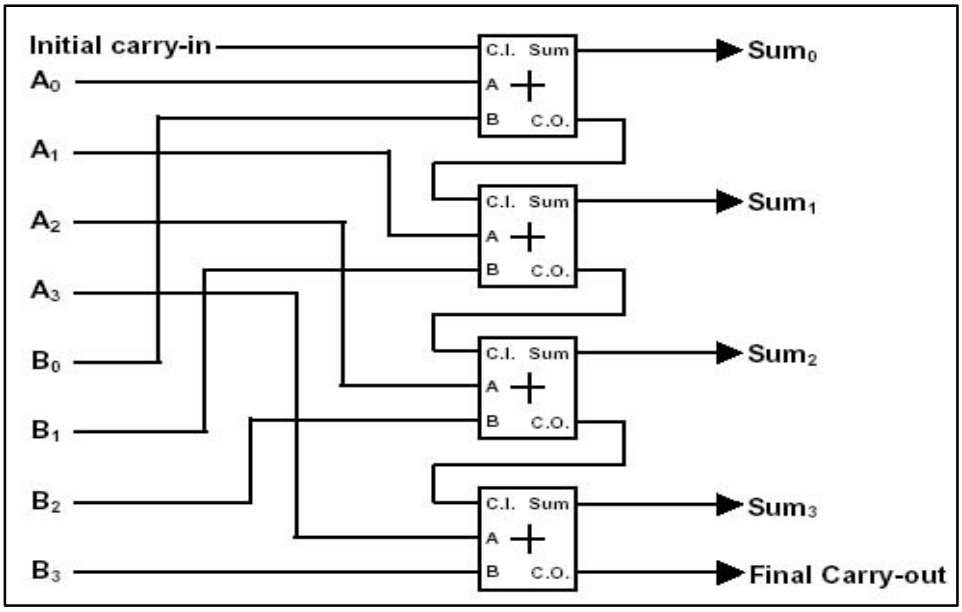
# 2

**Description**

Design and Develop the circuit for the following arithmetic function in VHDL Codes and Simulate it. Addition, Subtraction Multiplication (4 x 4 bits)

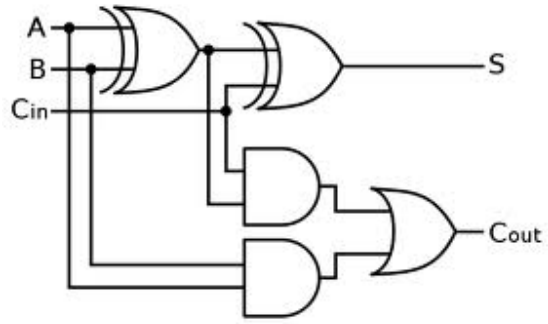
## 2.1 – Addition (Program for 4-bit addition using 4 bit Ripple adder)

It is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a $C_{in}$, which is the $C_{out}$ of the previous adder. This kind of adder is called a ripple-carry adder, since each carry bit "ripples" to the next full adder.

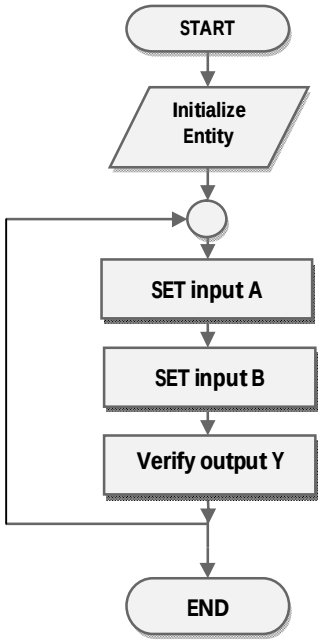


## Full Adder

The full-adder circuit adds three one-bit binary numbers (Cin, A, B) and outputs two one-bit binary numbers, a sum (S) and a carry (Cout).




## Flow Chart

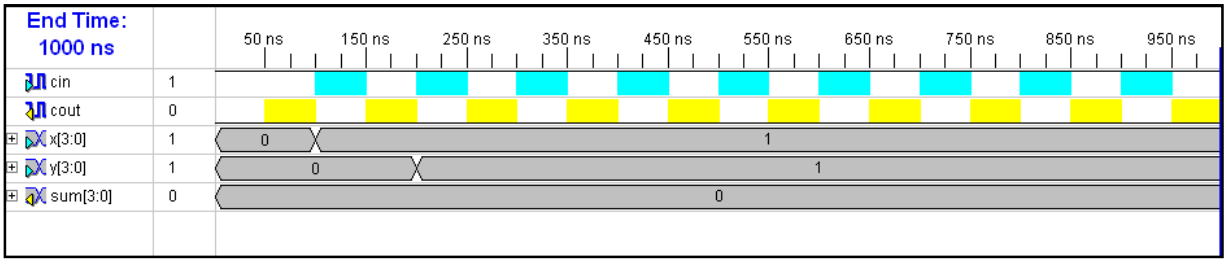


**Code:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adder is
   Port ( x: in  STD_LOGIC_VECTOR (3 downto 0);
          y : in  STD_LOGIC_VECTOR (3 downto 0);
          cin : in  STD_LOGIC;
          sum : out  STD_LOGIC_VECTOR (3 downto 0);
          cout : out  STD_LOGIC);
end adder;

architecture Behavioral of adder is

component fulladder
port(
      x,y,cin:in std_logic;
      sum,cout: out std_logic);
end component;
signal c: std_logic_vector(2 downto 0);

begin
a1:fulladder port map(x(0), y(0), cin, sum(0), c(0));
a2:fulladder port map(x(1), y(1), c(0), sum(1), c(1));
a3:fulladder port map(x(2), y(2), c(1), sum(2), c(2));
a4:fulladder port map(x(3), y(3), c(2), sum(3), cout);

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
port(
            x,y,cin:in std_logic;
            sum,cout: out std_logic);
end fulladder;

architecture comb of fulladder is
begin
sum <= x xor y xor cin;
cout <= (x and y) or (cin and (x xor y));
end comb;
```

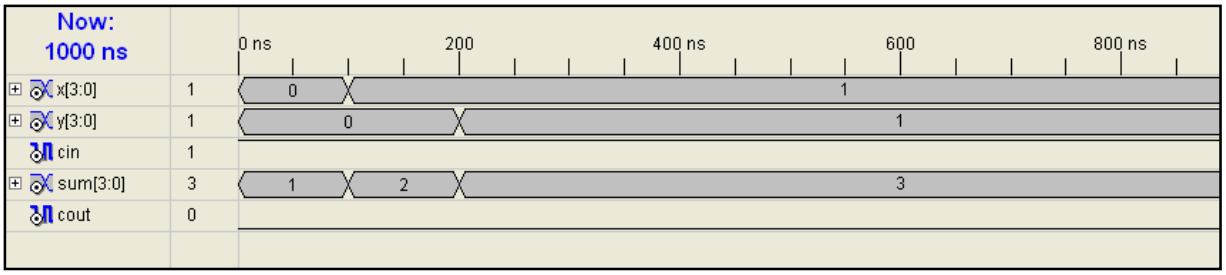
## Input Wave form

We give input x=’1’, y=’1’ and cin=’1’in waveform window



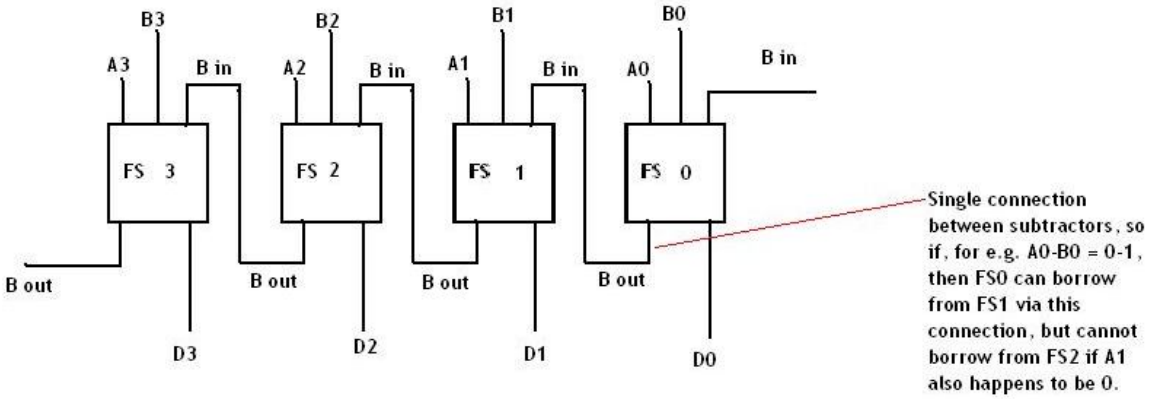

## Output Wave form

We give input x=’1’, y=’1’ and cin=’1’ so we get sum=’3’waveform window



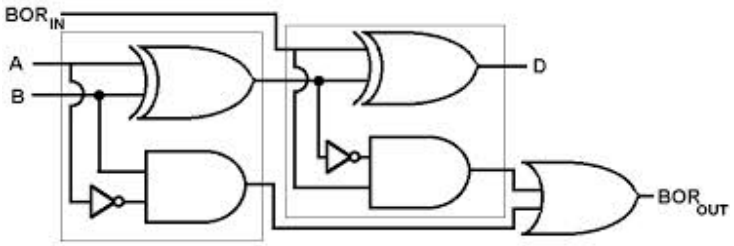
## 2.2- Subtraction (Program for 4-bit subtraction using arithmetic operator)

It is possible to create a logical circuit using multiple full subtractor to subtract N-bit numbers. Each full subtractor inputs a $B_{in}$, which is the $B_{out}$ of the previous subtractor . This kind of subtractor is called a ripple-carry subtractor, since each Borrow bit "ripples" to the next full subtractor.

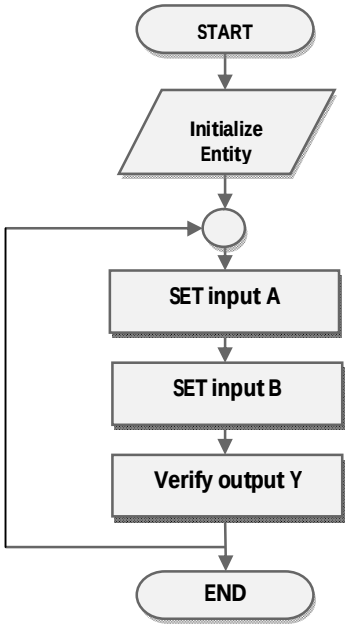



### Full Subtractor

A combinational circuit which performs the subtraction of three input bits is called full subtractor. The three input bits include two significant bits and a previous borrow bit.



➢ **Flow Chart**

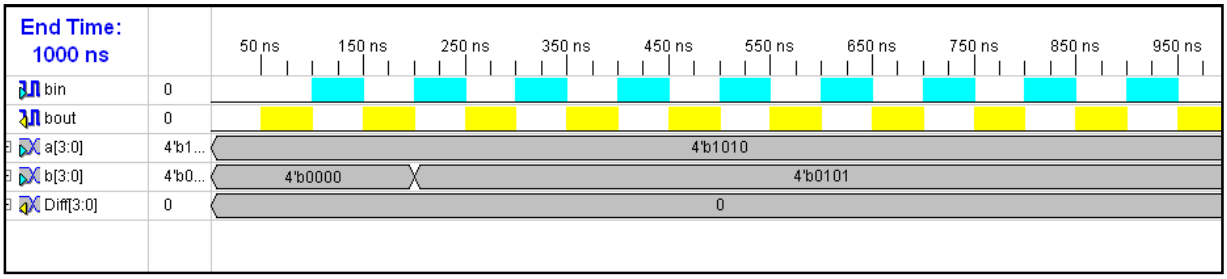

START
Initialize Entity
SET input A
SET input B
Verify output Y
END

## Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sub is
   Port ( a: in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          bin : in  STD_LOGIC;
          Diff : out  STD_LOGIC_VECTOR (3 downto 0);
          bout : out  STD_LOGIC);
end  sub;

architecture Behavioral of sub is
component fullsub
port(
       a,b,bin:in std_logic;
       Diff,bout: out std_logic);
end component;
signal D: std_logic_vector(2 downto 0);
begin
x1:fullsub port map(a(0), b(0), bin, Diff(0), D(0));
x2:fullsub port map(a(1), b(1), D(0), Diff(1), D(1));
x3:fullsub port map(a(2), b(2), D(1), Diff(2), D(2));
x4:fullsub port map(a(3), b(3), D(2), Diff(3), Bout);

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;

entity fullsub is
port(
       a,b,bin:in std_logic;
       Diff,bout: out std_logic);
end fullsub;

architecture comb of fullsub is
begin
Diff <= a xor b xor bin;
bout <= ((not a )and b) or ((not bin )and (a xor b)) ;

end comb;
```

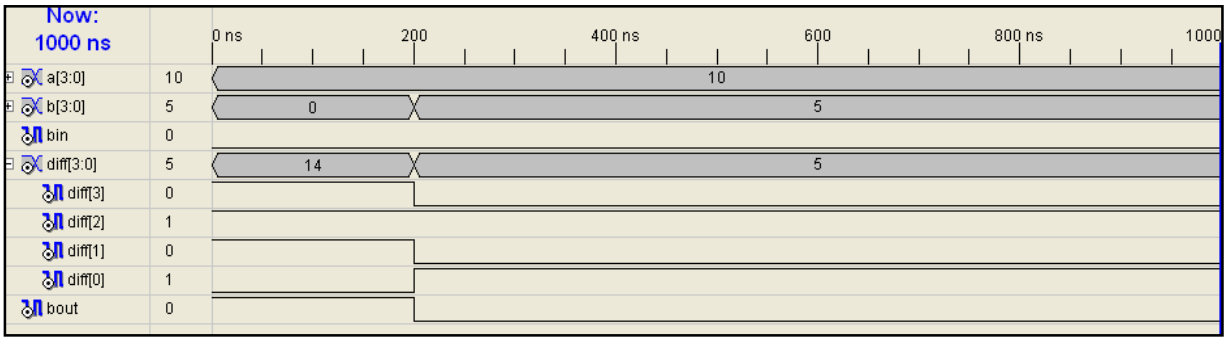
## Input Waveform

We give input a=’1010’, b=’0101’ in waveform window




## Output Waveform

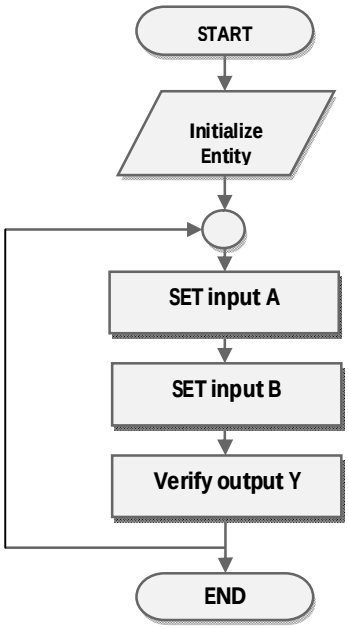
We get output in simulation window( according to that subtraction Operation)



## 2.3 – Multiplication (Program for simple 4x4 multiplications using arithmetic operator)

Consider the multiplication of two numbers as 4 x4 a * b, where a and b are 4 bit numbers and the output of multiplication is taken in y as 8 bit number.

### ➢ Flow Chart

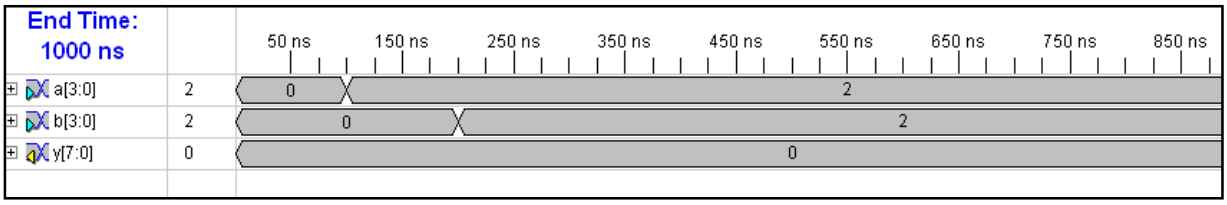


## Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity codef is
   Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          y : out  STD_LOGIC_VECTOR (7 downto 0));
end codef;

architecture Behavioral of codef is

begin

y <= a * b;
end Behavioral;
```

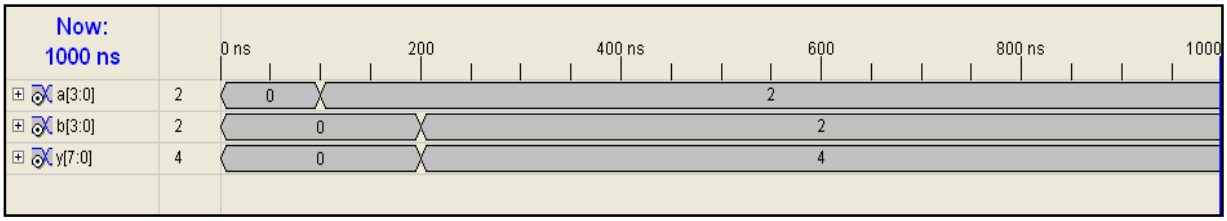
**Input Waveform**

We give input a =’2’& b =’2’ .




**Output Waveform**

We give input a =’2’& b =’2’ .so, we get output y=’4’ according to that multiplication operation.



## SIMULATION OF VHDL CODE FOR MULTIPLEXER

# 3

**Description**

Design and develop a 2 bit multiplexer and port map the same for developing up to 8 bit multiplexer.

### Multiplexer

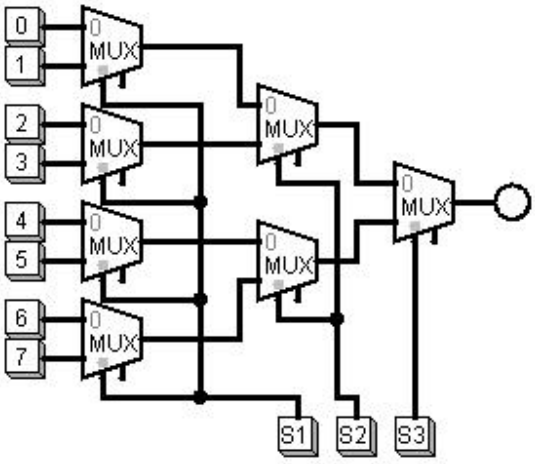
A multiplexer is a combinatorial circuit that is given a certain number (usually a power of two) data inputs, let us say $2^n$, and $n$ address inputs used as a binary number to select one of the data inputs. The multiplexer has a single output, which has the same value as the selected data input.

In other words, the multiplexer works like the input selector of a home music system. Only one input is selected at a time, and the selected input is transmitted to the single output. While on the music system, the selection of the input is made manually, the multiplexer chooses its input based on a binary number, the address input.

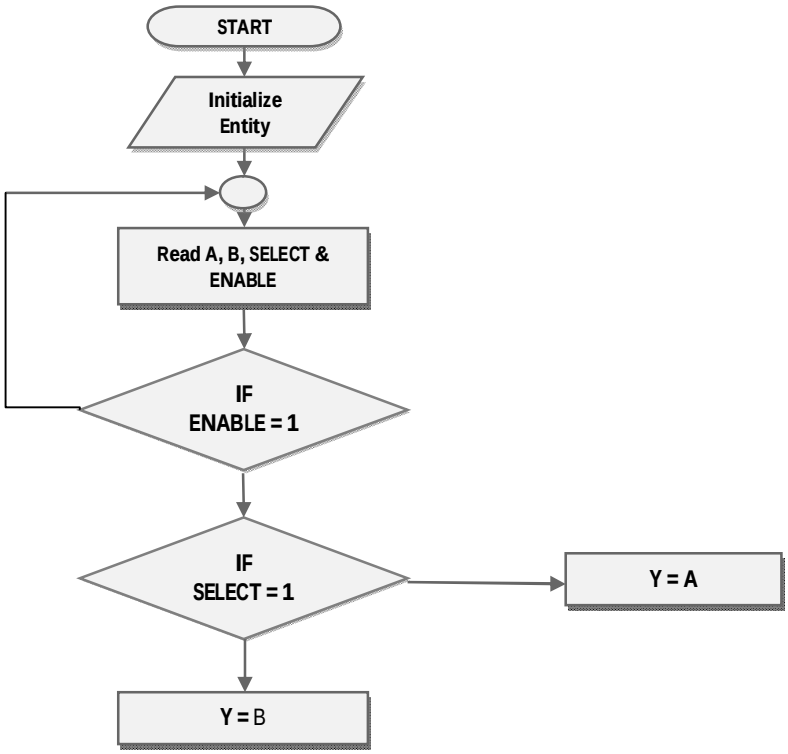
The truth table for a multiplexer is huge for all but the smallest values of $n$. We therefore use an abbreviated version of the truth table in which some inputs are replaced by `-' to indicate that the input value does not matter.

Here is such an abbreviated truth table for $n = 3$. The full truth table would have $2^{(3 + 23)} = 2048$ rows.

**2:1 mux to construct 8:1 mux**




**Flow chart**



## Code Listing

### 2:1 multiplexer program

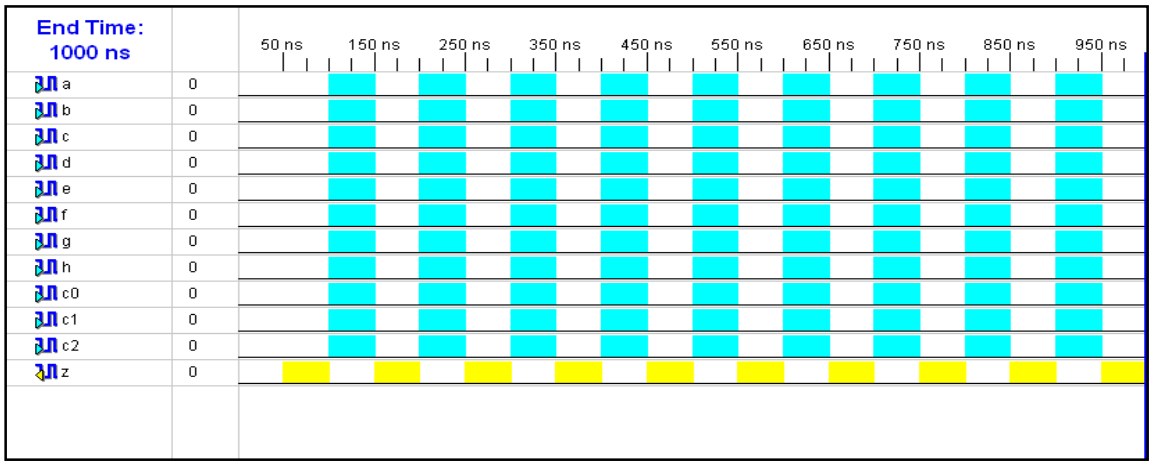
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux is
    Port ( x : in  STD_LOGIC;
           y : in  STD_LOGIC;
           sel : in  STD_LOGIC;
           z : out  STD_LOGIC);
end mux;

architecture Behavioral of mux is
begin
process (x,y,sel)
begin
if(sel='0') then
z<=x;
elsif (sel='1') then
z<=y;
end if;
end process;
```

**8:1 mux port map program**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity multiplexer is
    Port (
             a, b,c,d,e,f,g,h : in  STD_LOGIC ;
     c0,c1,c2: in std_logic;
                  z : out  STD_LOGIC );
end multiplexer;
architecture Behavioral of multiplexer is
component mux
port(
 x:in std_logic;
 y: in std_logic;
 sel: in std_logic;
 z: out std_logic
 );
 end component ;
 signal z1,z2,z3,z4,z5,z6 : std_logic :='0';
 begin
      signal z1,z2,z3,z4,z5,z6 : std_logic :='0';
 begin
mux1: mux port map (a, b, c0, z1);
mux2: mux port map (c, d, c0, z2);
mux3: mux port map (e, f, c0, z3);
mux4  mux port map (g, h, c0, z4);
mux5: mux port map (z1, z2, c1, z5);
mux6: mux port map (z3, z4, c1, z6);
mux7: mux port map (z5, z6, c2, z);

end Behavioral;
```

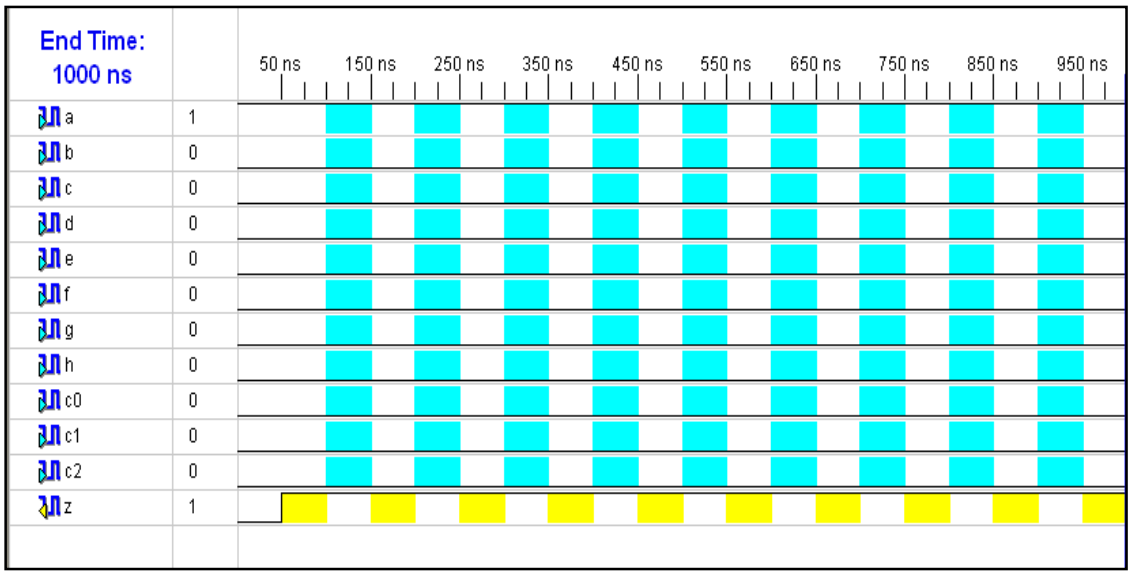
## Input Waveform

We give input a=’high’ in waveform window




## Output Waveform

We get output in simulation window (y=’high’ according to that multiplexer operation)



## SIMULATION OF VHDL CODE FOR DEMULTIPLEXER

3

**Description**

Design and develop an 8 output demultiplexer using truth table.

**Demultiplexer**

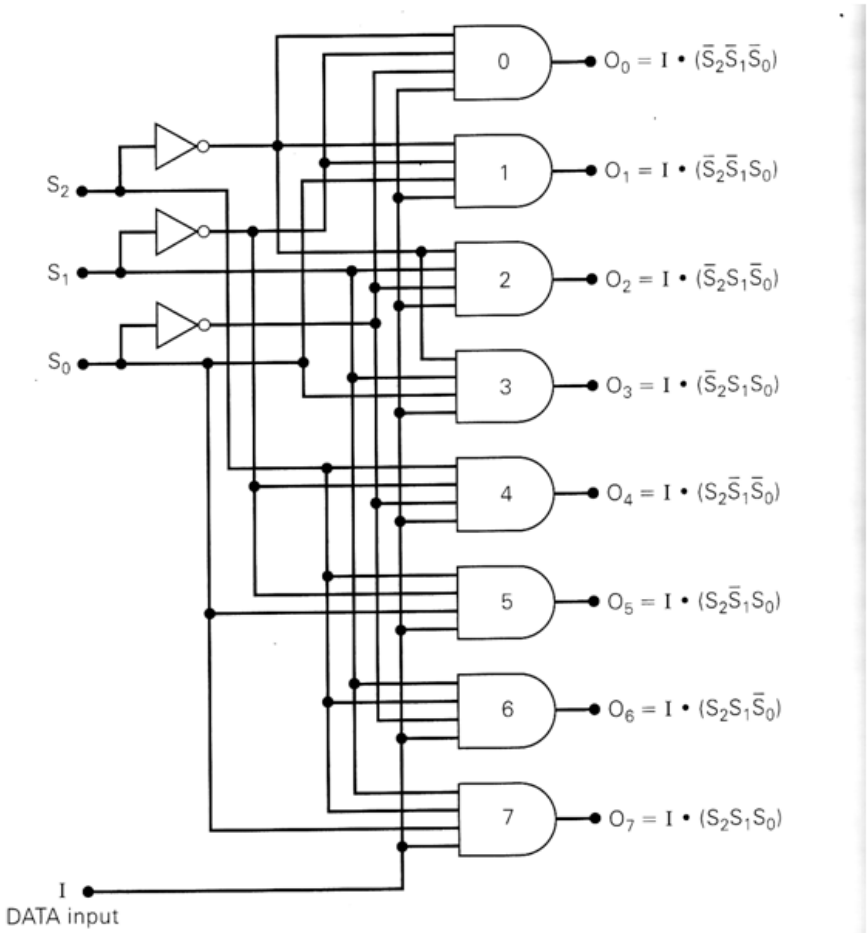
The demultiplexer is the inverse of the multiplexer, in that it takes a single data input and n address inputs. It has $2^n$ outputs. The address input determine which data output is going to have the same value as the data input. The other data outputs will have the value 0.

Here is an abbreviated truth table for the demultiplexer. We could have given the full table since it has only 16 rows, but we will use the same convention as for the multiplexer where we abbreviated the values of the data inputs.

| S2 | S1 | S0 | E | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | Y1 | Y0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Here is one possible circuit diagram for the demultiplexer:

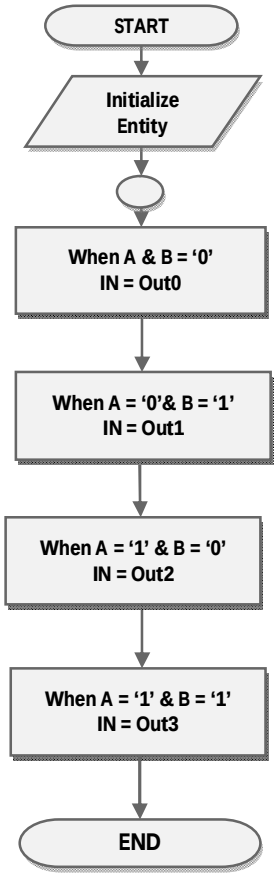
In this program a 1 x 8 de-multiplexer have two 1- bit inputs, a 3-bit select line and a 8- bit output. Additional control signals may be added such as enable. The output of the multiplexers depends on the level of the select line.




| SELECT code | | | OUTPUTS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | I |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | I | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | I | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | I | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | I | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | I | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | I | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | I | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Note: I is the data input

## Flow Chart

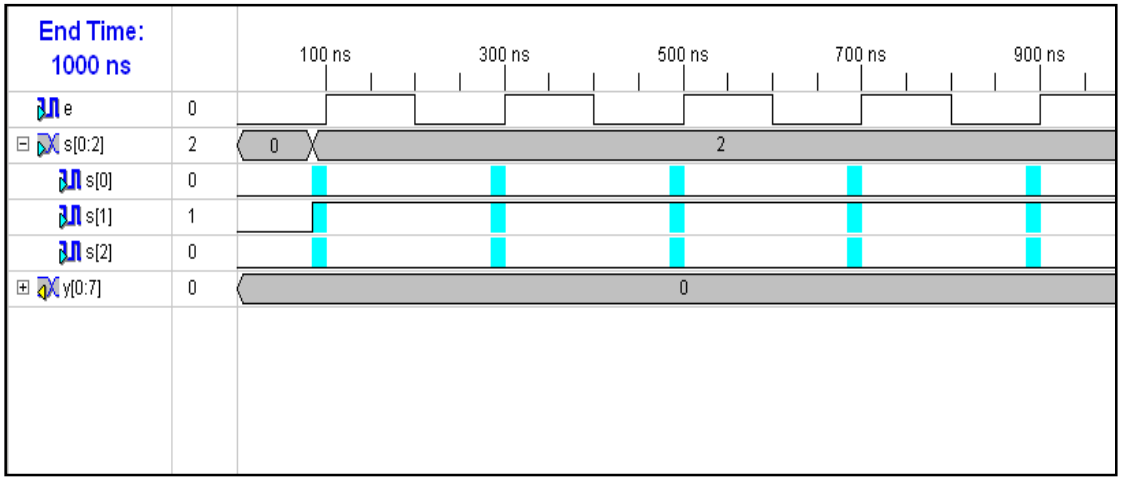

START
Initialize Entity
When A & B = '0' IN = Out0
When A = '0'& B = '1' IN = Out1
When A = '1' & B = '0' IN = Out2
When A = '1' & B = '1' IN = Out3
END

## Code Listing

```
library ieee;
use ieee.std_logic_1164.all;

entity demultiplexer is
port(I : in std_logic;
     s : in std_logic_vector(0 to 2);
     O : out std_logic_vector(0 to 7));
end demultiplexer;

architecture data of demultiplexer is

begin
    O(0) <= (I and not s(0) and not s(1) and not s(2));
    O(1) <= (I and s(0) and not s(1) and not s(2));
    O(2) <= (I and not s(0) and s(1) and not s(2));
    O(3) <= (I and s(0) and s(1) and not s(2));
    O(4) <= (I and not s(0) and not s(1) and s(2));
    O(5) <= (I and s(0) and not s(1) and s(2));
    O(6) <= (I and not s(0) and s(1) and s(2));
    O(7) <= (I and s(0) and s(1) and s(2));
end data;
```

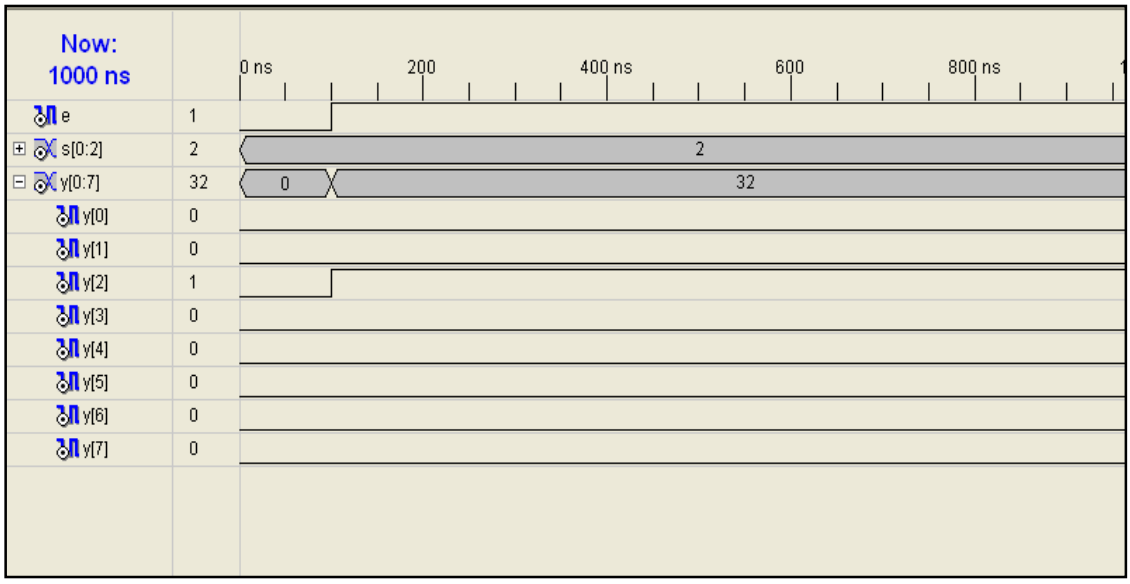
**Input Waveform**

We put enable as high in all condition and here we give input=’010’ in selection line(according to that selection line we get output)




**Output Waveform**

We get output in simulation window(according to that selection line)



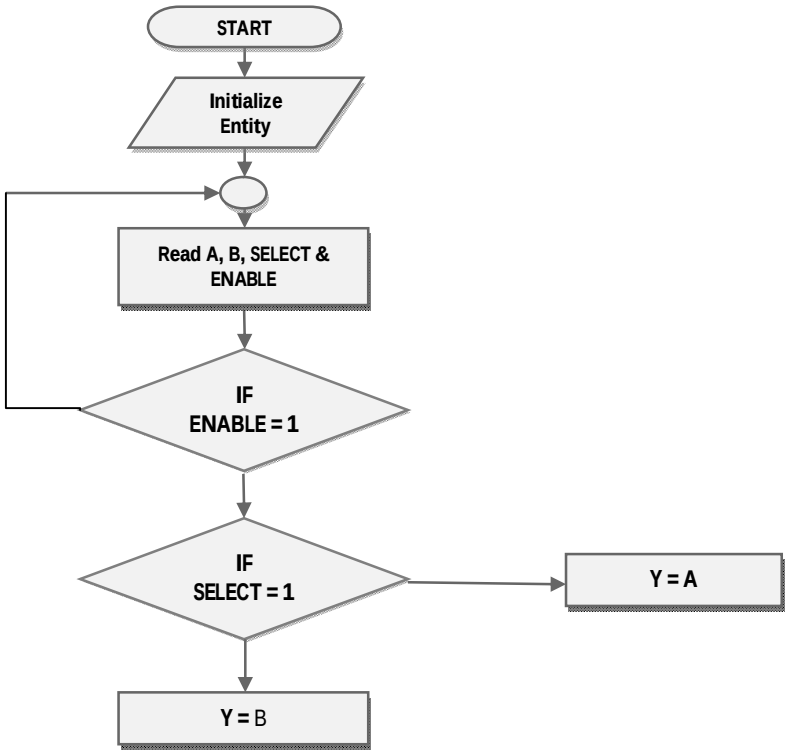
# VHDL IMPLEMENTATION OF MULTIPLEXER

# 5

## Description

Describe the code for a multiplexer and implement it in FPGA kit in which switches are connected for select input and for data inputs a LED is connected to the output.

## Flow Chart



**PIN Description:**

| I/O Pins | A | B | C | D | SEL0 | SEL1 | OUTPUT |
|---|---|---|---|---|---|---|---|
| FPGA LOC | P86 | P87 | P89 | P90 | P92 | P93 | P105 |

### Code Listing

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mux_gate is

port (A,B,C,D: in STD_LOGIC;
      SEL     : in STD_LOGIC_vector (1 downto 0);
      Output    : out STD_LOGIC);
end mux_gate;

architecture behav of mux_gate is
begin
process (SEL,A,B,C,D)
begin
case SEL is
when "00"  => Output <= A;
when "01"  => Output <= B;
when "10"  => Output <= C;
when "11"  => Output <= D;
when others => null;
end case;
end process;
end behave;
```

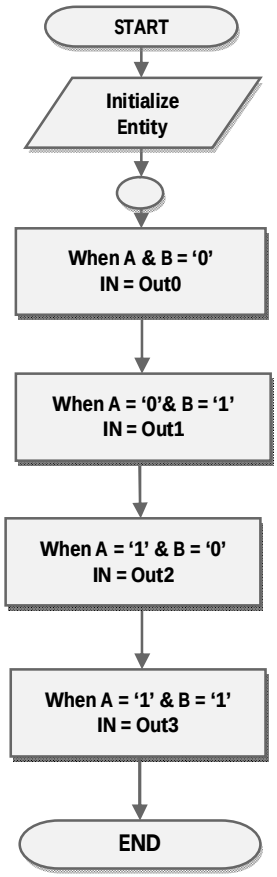
# VHDL IMPLEMENTATION OF DEMULTIPLEXER

6

## Description

Switches are connected for select inputs and a data input, Eight LEDs are connected to the output of the circuit.

## Flow Chart



**PIN Description:**

| I/O PINS | E | S(0) | S(1) | S(2) | Y(0) | Y(1) | Y(2) | Y(3) | Y(4) | Y(5) | Y(6) | Y(7) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FPGA LOC | P86 | P87 | P89 | P90 | P97 | P98 | P99 | P100 | P102 | P103 | P104 | P105 |

## Code Listing

```
library ieee;
use ieee.std_logic_1164.all;

entity demultiplexer is
port(e : in std_logic;
     s : in std_logic_vector(0 to 2);
     y : out std_logic_vector(0 to 7));
end demultiplexer;

architecture data of demultiplexer is

begin
    y(0) <= (e and not s(0) and not s(1) and not s(2));
    y(1) <= (e and s(0) and not s(1) and not s(2));
    y(2) <= (e and not s(0) and s(1) and not s(2));
    y(3) <= (e and s(0) and s(1) and not s(2));
    y(4) <= (e and not s(0) and not s(1) and s(2));
    y(5) <= (e and s(0) and not s(1) and s(2));
    y(6) <= (e and not s(0) and s(1) and s(2));
    y(7) <= (e and s(0) and s(1) and s(2));
end data;
```

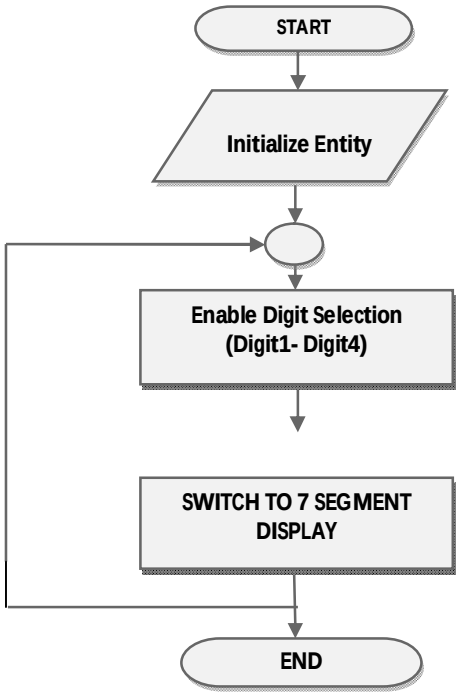
# VHDL IMPLEMENTATION OF 7 SEGMENT DECODER

7

## Description

Develop Boolean expression for 4 input variables and 7 output variables. Design and develop seven segment decoder in VHDL for 7 equations. A seven segment display is connected to the output of the circuit. Four switches are connected to the input. The 4 bit input is decoded to 7 segment equivalent.

## Flow Chart



**PIN Description:**

| I/O PINS | I0 | I1 | I2 | I3 | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FPGA LOC | P86 | P87 | P89 | P90 | P23 | P24 | P25 | P26 | P27 | P28 | P30 |

| SEL0 | SEL1 | SEL2 | SEL3 | SEL4 | SEL5 | SEL6 | SEL7 |
|---|---|---|---|---|---|---|---|
| P12 | P13 | P14 | P15 | P17 | P18 | P20 | P21 |

**Look Up Table:**

| I3 | I2 | I1 | I0 | G | F | E | D | C | B | A |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

**Code Listing**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity seven_seg is
    Port ( I3,I2,I1,I0 : in  STD_LOGIC;
                  sel : out std_logic_vector (3 downto 0);
           A,B,C,D,E,F,G : out  STD_LOGIC);
end seven_seg;

architecture Behavioral of seven_seg is
SIGNAL X0,X1,X2,X3,X4,X5,X6,X7,X8,X9:STD_LOGIC;
begin
sel <= "11111111";
X0<= (NOT I3 AND NOT I2 AND NOT I1 AND NOT I0);
X1<= (NOT I3 AND NOT I2 AND NOT I1 AND I0);
X2<= (NOT I3 AND NOT I2 AND I1 AND NOT I0);
X3<= (NOT I3 AND NOT I2 AND I1 AND I0);
X4<= (NOT I3 AND I2 AND NOT I1 AND NOT I0);
X5<= (NOT I3 AND I2 AND NOT I1 AND I0);
X6<= (NOT I3 AND I2 AND I1 AND NOT I0);
X7<= (NOT I3 AND I2 AND I1 AND I0);
X8<= (I3 AND NOT I2 AND NOT I1 AND NOT I0);
X9<= (I3 AND NOT I2 AND NOT I1 AND I0);

A <= X1 OR X4;
B <= X5 OR X6;
C <= X2;
D <= X1 OR X4 OR X7 OR X9;
E <= X1 OR X3 OR X4 OR X5 OR X7 OR X9;
F <= X1 OR X2 OR X3 OR X7;
G <= X0 OR X1 OR X7;
end Behavioral;
```

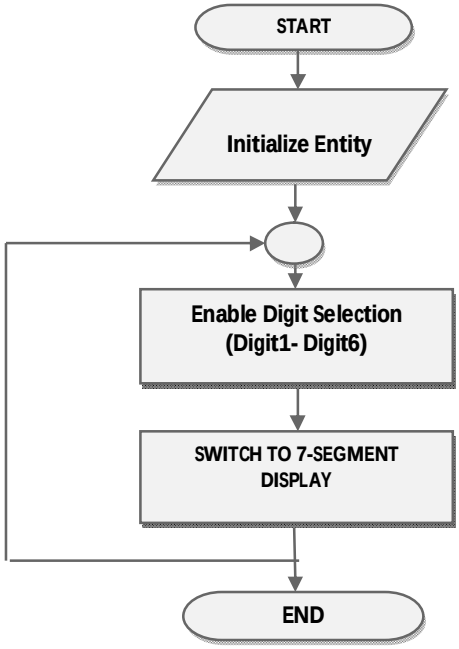
# VHDL IMPLEMENTATION OF 7 SEGMENT DECODER BY LUT

8

## Description

Develop a 7 segment decoder using Look up table. Describe the seven segment decoder in VHDL using developed Look up table. A seven segment display is connected to the output of the circuit. Four switches are connected to the input. The 4 bit input is decoded into 7 segment equivalent.

## Flow Chart



**Look Up Table:**

| I3 | I2 | I1 | I0 | G | F | E | D | C | B | A |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

**PIN Description:**

| I/O PINS | I0 | I1 | I2 | I3 | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FPGA LOC | P86 | P87 | P89 | P90 | P23 | P24 | P25 | P26 | P27 | P28 | P30 |

| SEL0 | SEL1 | SEL2 | SEL3 | SEL4 | SEL5 | SEL6 | SEL7 |
|---|---|---|---|---|---|---|---|
| P12 | P13 | P14 | P15 | P17 | P18 | P20 | P21 |

**Code Listing**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity SEVEN_SEG_LUT is
    Port ( I : in  STD_LOGIC_VECTOR (3 downto 0);
           SEL : out  STD_LOGIC_VECTOR (7 downto 0);
           Y : out  STD_LOGIC_VECTOR (6 downto 0));
end SEVEN_SEG_LUT;

architecture Behavioral of SEVEN_SEG_LUT is

begin

SEL <= "11111111";
process (I)
BEGIN
case  I is
when "0000"=> Y <="1000000";  -- '0'
when "0001"=> Y <="1111001";  -- '1'
when "0010"=> Y <="0100100";  -- '2'
when "0011"=> Y <="0110000";  -- '3'
when "0100"=> Y <="0011001";  -- '4'
when "0101"=> Y <="0010010";  -- '5'
when "0110"=> Y <="0000010";  -- '6'
when "0111"=> Y <="1111000";  -- '7'
when "1000"=> Y <="0000000";  -- '8'
when "1001"=> Y <="0011000";  -- '9'
when others=> Y <="1111111";
end case;

end process;

end Behavioral;
```

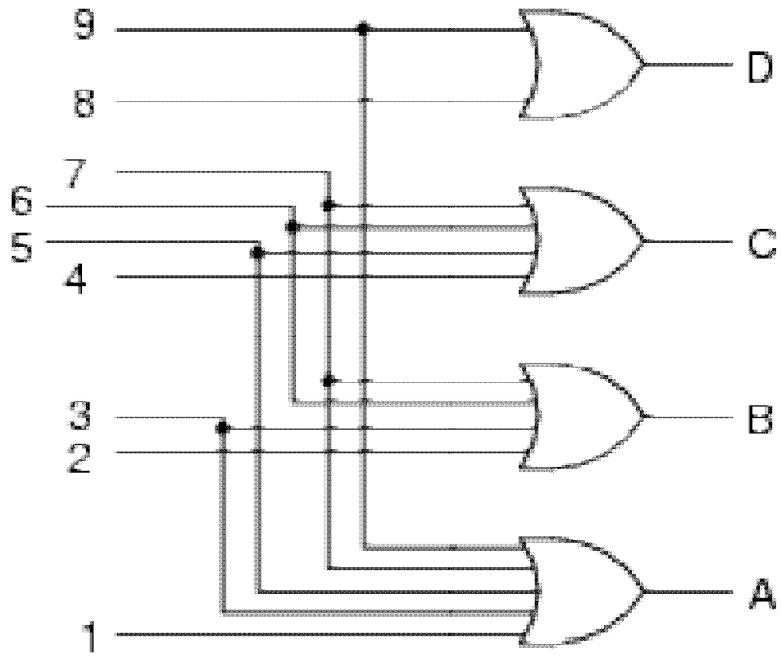
## VHDL IMPLEMENTATION OF ENCODER

**9**

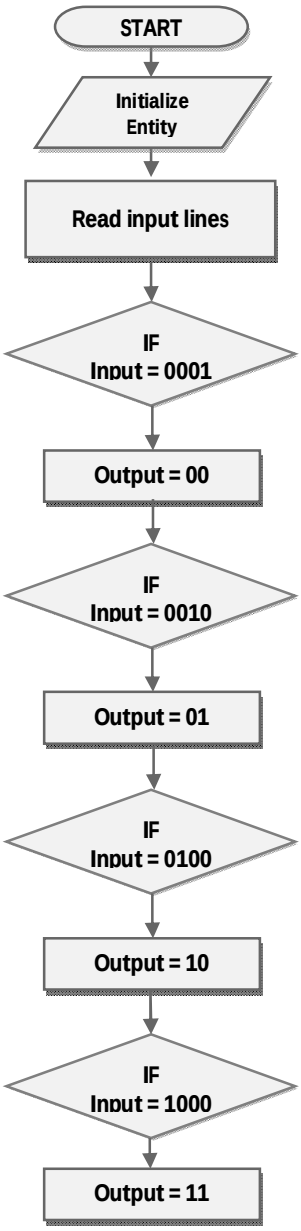
### *Encoder:*

Design and develop HDL code for decimal (Octal) to BCD encoder. There will be10 input switches (or 8 switches) and 4 LEDs in the FPGA kit. The input given from switches and it is noted that any one of the switch is active. The binary equivalent for the corresponding input switch will be glowing in the LED as output

**Logical Diagram**



## Flow Chart


START
Initialize Entity
Read input lines
IF Input = 0001
Output = 00
IF Input = 0010
Output = 01
IF Input = 0100
Output = 10
IF Input = 1000
Output = 11

**PIN Description:**

| I/O PINS | X(0) | X(1) | X(2) | X(3) | X(4) | X(5) | X(6) | X(7) | A | B | C | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FPGA LOC | P86 | P87 | P89 | P90 | P92 | P93 | P95 | P96 | P97 | P98 | P99 | P100 |

**Code Listing**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( x : in std_logic_vector(7 downto 0);
       a,b,c,d : out std_logic;);
end first;
architecture Behavioral of first is

begin
a <= x(1) or x(3) or x(5) or x(7) or x(9);
b <= x(2) or x(3) or x(6) or x(7) ;
c <= x(4) or x(5) or x(6) or x(7) ;
d <= x(9) or x(8) ;
end Behavioral;
```

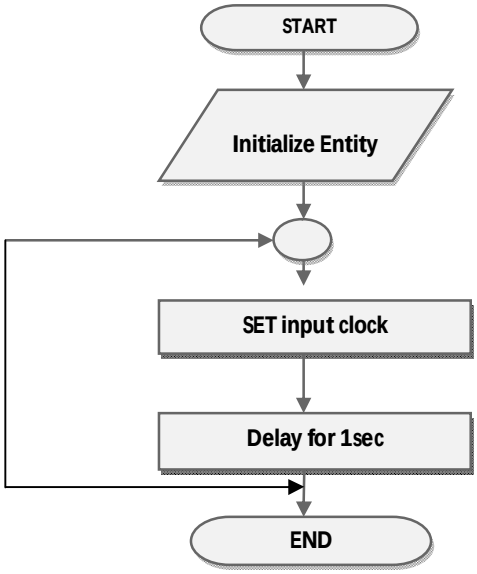
## SIMULATION OF VHDL CODE FOR DELAY

**10**

### Description

Develop a VHDL code for making a delayed output for 1second or 2 seconds by assuming clock frequency provided in the FPGA Kit. Simulate same code to get a delayed waveform.

### Flow Chart

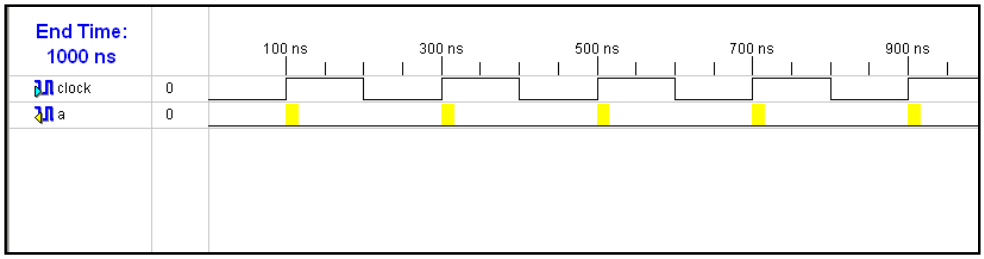


### Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( clock : in std_logic;
       a     : out std_logic);
end first;

architecture Behavioral of first is

begin
process(clock)
variable i : integer := 0;
begin
if clock'event and clock = '1' then
      if i <= 50000000 then
            i := i + 1;
            a <= '1';
            elsif i > 50000000 and i < 100000000 then
            i := i + 1;
            a <= '0';
            elsif i = 100000000 then
            i := 0;
      end if;
end if;
end process;
end Behavioral;
```

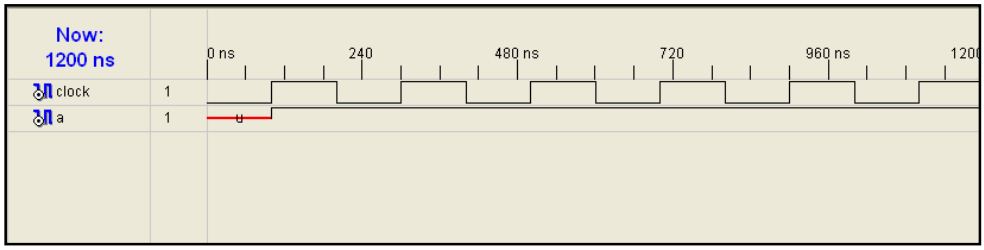
## Input Waveform

We give input in waveform window (clk=’1’)




## Output Waveform

We get output in simulation waveform (a=’delayed signal’)



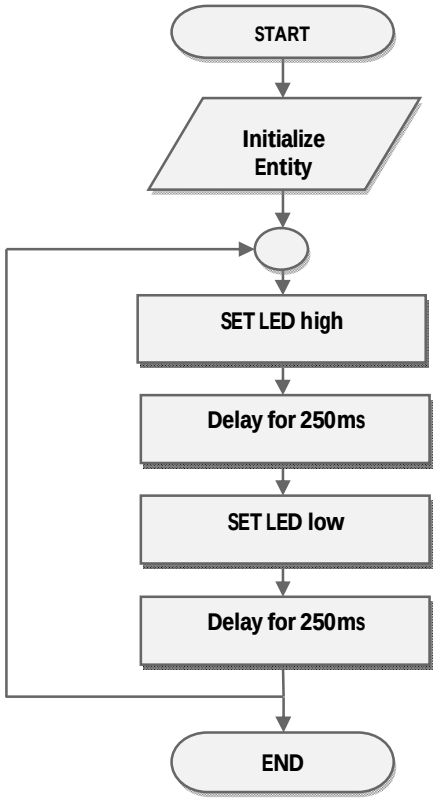
# VHDL IMPLEMENTATION FOR BLINKING A LED

11

## Description

Develop a VHDL Code for delay and verify by simulating it. This delay output is connected to LED. Delay is adjusted such away LED blinks for every 1 or 2 seconds.

## Flow Chart



## PIN Description:

| I/O PINS | CLK | LED |
|---|---|---|
| FPGA LOC | P55 | P97 |

**Code Listing**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity first is
port ( clock : in std_logic;

            LED     : out std_logic);
end first;

architecture Behavioral of first is

begin
process(clock)
variable i : integer := 0;
begin
if clock'event and clock = '1' then
if i <= 50000000 then
i := i + 1;
LED <= '0';
elsif i > 50000000 and i < 100000000 then
i := i + 1;
LED <= '1';
elsif i = 100000000 then
i := 0;
end if;
end if;
end process;
end Behavioral;
```

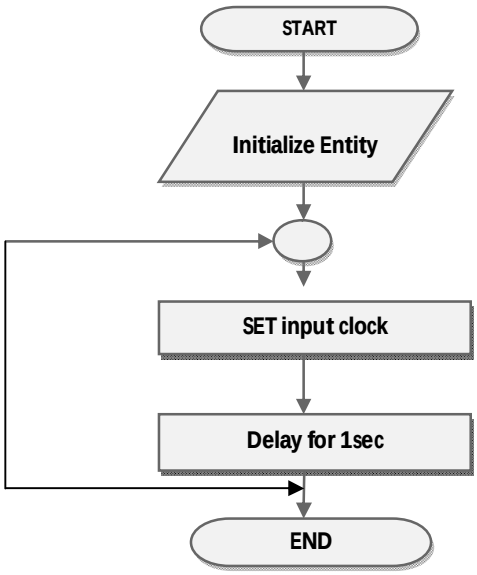
## SIMULATE A VHDL TEST BENCH CODE FOR TESTING A GATE

**12**

### Description

Develop a VHDL test bench code for testing any one of the simple gate. Simulate the test bench code in the HDL software.

### Flow Chart



**Code Listing (AND gate program)**

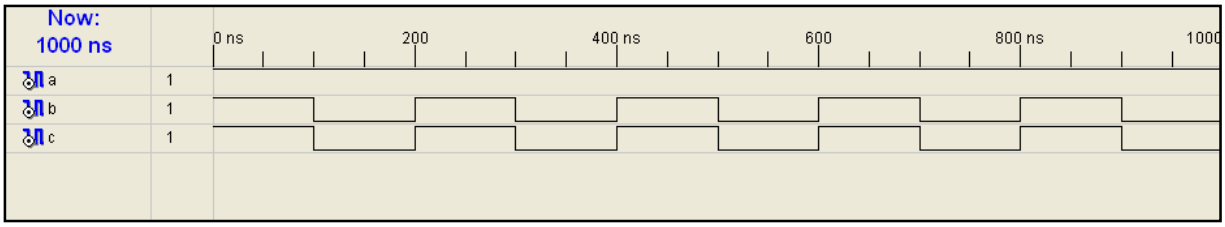
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity gate is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : out  STD_LOGIC);
end gate;

architecture Behavioral of gate is

begin
c <= a and  b;

end Behavioral;
```

***Testbench code***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY code_vhd IS
END code_vhd;

ARCHITECTURE behavior OF code_vhd IS

	-- Component Declaration for the Unit Under Test (UUT)
	COMPONENT cd
	PORT(
		a : IN std_logic;
		b : IN std_logic;
		c : OUT std_logic
		);
	END COMPONENT;

	--Inputs
	SIGNAL a :  std_logic := '0';
	SIGNAL b :  std_logic := '0';

	--Outputs
	SIGNAL c :  std_logic;

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: cd PORT MAP(
		a => a,
		b => b,
		c => c
	);

	tb : PROCESS
	BEGIN
		a<='1';
		b<='1';
		-- Wait 100 ns for global reset to finish
		wait for 100 ns;
		a<='1';
		b<='0';
		-- Place stimulus here

		wait for 100 ns; -- will wait forever
	END PROCESS;

END;
```

**Output Waveform**

We get output in simulation window (c=’1’ according to that gate operation)



\

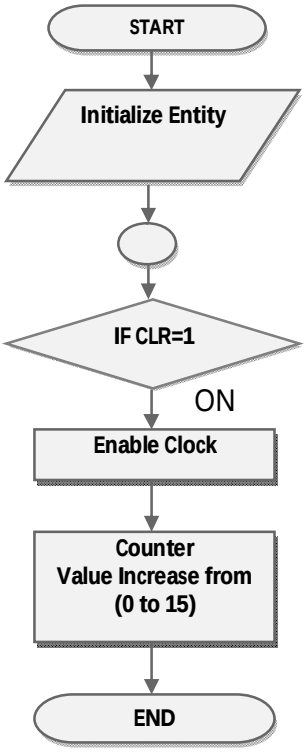
# VHDL IMPLEMENTATION FOR BLINKING A ARRAY OF LEDS

13

## Description

Design and develop a VHDL Code for 4 bit binary up counter. Four LEDs are connected at the output of the counter. The counter should up for every one seconds.

## Flow Chart



**PIN Description:**

| I/O PINS | CLOCK | RST | Q(3) | Q(2) | Q(1) | Q(0) |
|---|---|---|---|---|---|---|
| FPGA LOC | P55 | P86 | P97 | P98 | P99 | P100 |

**Code Listing**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( clock : in std_logic;
      rst:in std_logic;
      q     : out std_logic_vector(3 downto 0);
end first;

architecture Behavioral of first is
signal tmp: std_logic_vector(3 downto 0):="0000";
begin
process(clock ,rst)
variable i : integer := 0;
begin
if (rst='1') then
tmp <= "0000";
elsif clock'event and clock = '1' then
      if i <= 50000000 then
      i := i + 1;
      elsif i = 50000001 then
      i := 0;
      tmp <= tmp+1;
end if;
end if;
end process;
q<= tmp;
end Behavioral;
```

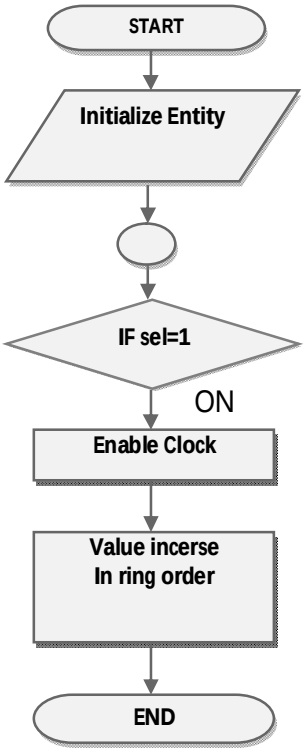
# VHDL IMPLEMENTATION OF A SPELLER WITH AN ARRAY OF LEDS

# 14

## Description

Design and develop VHDL Code for a 5 bit Johnson ring counter 4 bit The LEDs are connected at the output of the counter. The speller should work for every one seconds.

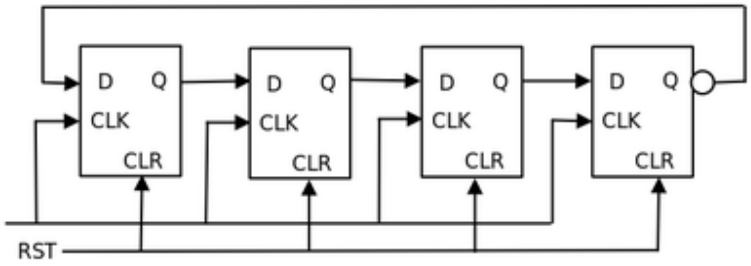
## Flow Chart




## PIN Description:

| I/O PINS | CLOCK | RST | Q(4) | Q(3) | Q(2) | Q(1) | Q(0) |
|---|---|---|---|---|---|---|---|
| FPGA LOC | P55 | P86 | P97 | P98 | P99 | P100 | P102 |

**Johnson Ring Counter**




**CODE Listing**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( clk : in std_logic;
       Reset: in std_logic;
       output : out std_logic_vector(4 downto 0));
end first;

architecture Behavioral of first is
signal temp: std_logic_vector(4 downto 0):=”00000”;
begin
process(clk, Reset)
variable i,k : integer := 0;
begin
if Reset = ‘1’ then
temp <= “00000”;
elsif clk'event and clk = '1' then
if i < 50000000 then
i := i + 1;
elsif i = 50000000 then
temp(1) <= temp(0);
temp(2) <= temp(1);
temp(3) <= temp(2);
temp(4) <= temp(3);
temp(0) <= not temp(4);
i:=0;
end if;
end if;
end process;
Reset <= temp;
End Behavioral;
```

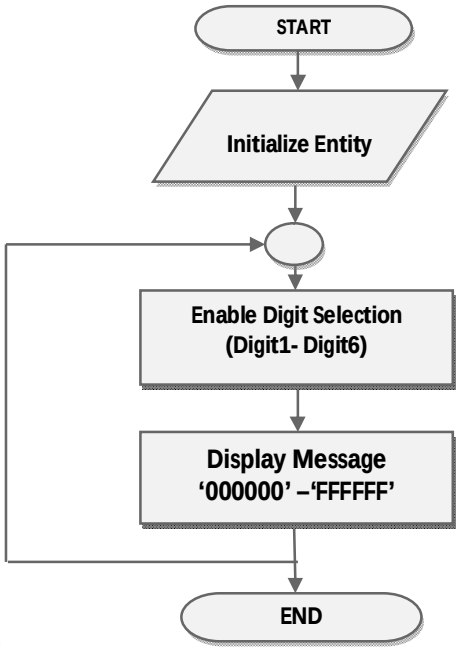
## VHDL IMPLEMENTATION OF 7 SEGMENT DISPLAY

**15**

### Description

Design and develop a seven segment decoder in VHDL. Design and develop a 4 bit BCD counter, the output of the counter is given to seven segment decoder. A seven segment display is connected to the output of the decoder. The display shows 0,1, 2.. 9 for every one second

### Flow Chart




### PIN Description:

| I/O PINS | CLK | Y(0) | Y(1) | Y(2) | Y(3) | Y(4) | Y(5) | Y(6) | Y(7) |
|---|---|---|---|---|---|---|---|---|---|
| FPGA LOC | P55 | P23 | P24 | P25 | P26 | P27 | P28 | P30 | P31 |

| SEL0 | SEL1 | SEL2 | SEL3 | SEL4 | SEL5 | SEL6 | SEL7 |
|---|---|---|---|---|---|---|---|
| P12 | P13 | P14 | P15 | P17 | P18 | P20 | P21 |

**Code Lisitng**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity object_co  is

port(clk    : in  std_logic;
     y      : out std_logic_vector(7 downto 0);
      sel     : out std_logic_vector(7 downto 0)
     );
end object_co;

architecture beav of object_co  is
type state is
(state0,state1,state2,state3,state4,state5,state6,state7,state8,
state9);
signal next_state,ps: state := state0;
begin
sel <= "11111111";
process(clk,next_state)
variable i : integer := 0 ;
begin
if clk'event and clk = '1' then
if i <= 100000000 then
i := i + 1;
elsif i > 100000000 then
i := 0 ;
next_state <= ps ;
end if;
if next_state = state0 then
y <= x"c0" ;
ps <= state1;
```

```
elsif next_state = state1 then
y <= x"f9";
ps <= state2;
elsif next_state = state2 then
y <= X"a4";
ps <= state3;
elsif next_state = state3 then
y <= X"b0";
ps <= state4;
elsif next_state = state4 then
y <= X"99";
ps <= state5;
elsif next_state = state5 then
y <= X"92";
ps <= state6;
elsif next_state = state6 then
y <= X"82";
ps <= state7;
elsif next_state = state7 then
y <= X"f8";
ps <= state8;
elsif next_state = state8 then
y <= X"80";
ps <= state9;
elsif next_state = state9 then
y <= X"98";
ps <= state0;
end if;
end if;
end process;
end beav;
```

# *Getting Started with Xilinx ISE*

## *(Tutorial)*

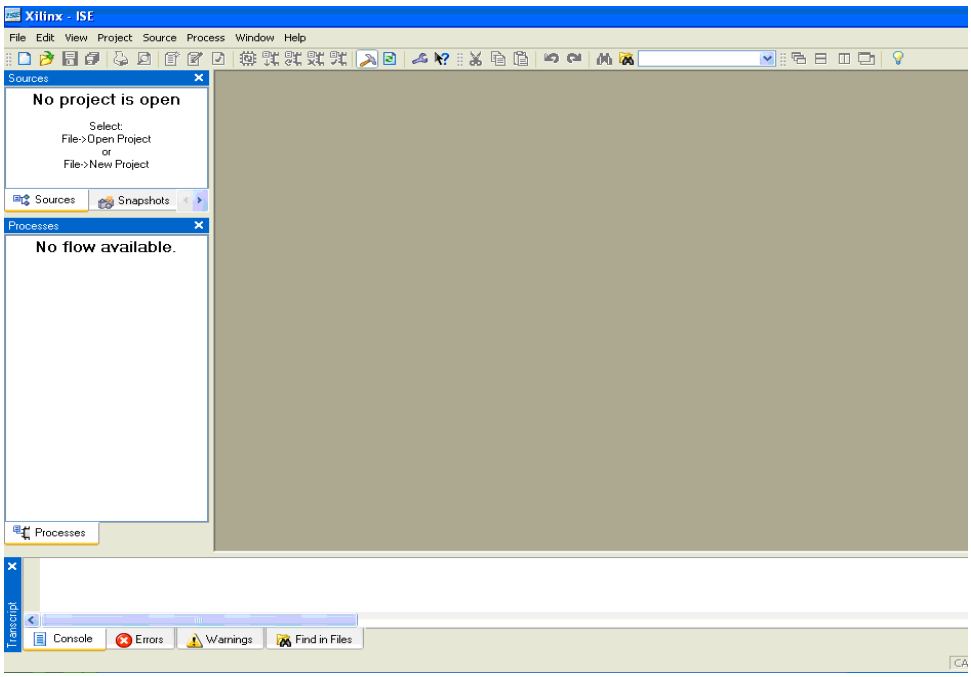
## *9 - Getting Started with Xilinx ISE*

### Starting the ISE 8.1 software

To start ISE, double – click the desktop icon,


Or start ISE from the start menu by selecting:

Start → All Programs → Xilinx ISE 8.1i → Project Navigator.




### Create a New Project

Create a new ISE project which will target the FPGA device on the Spartan-3 FPGASP3 Kit.

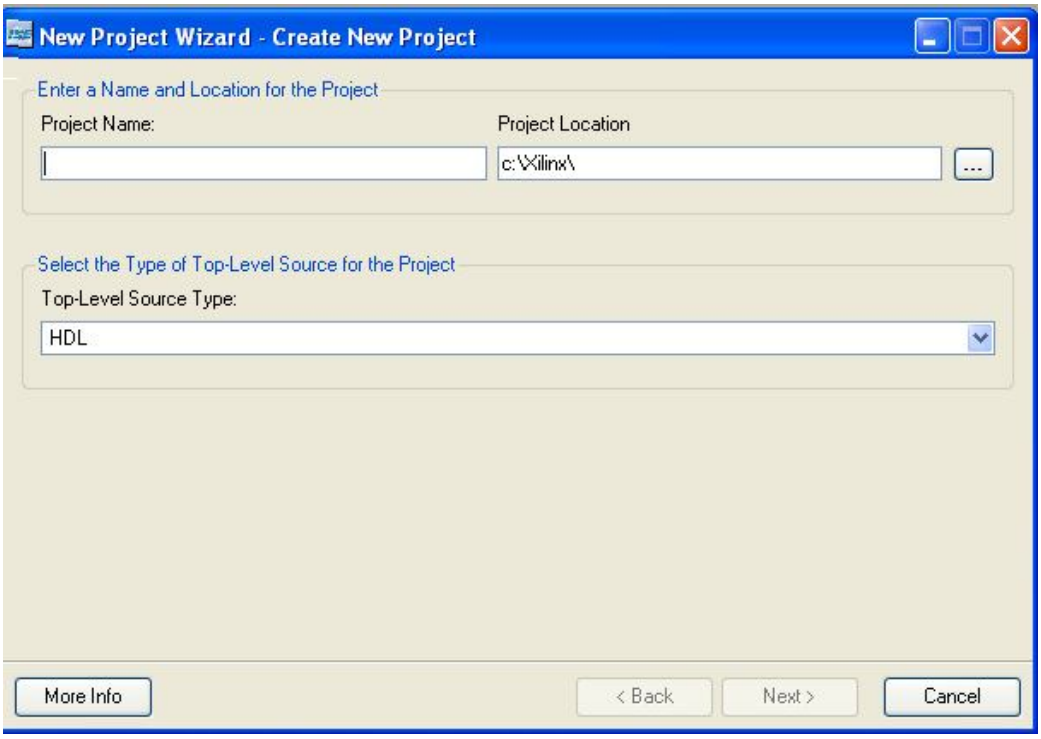
**To create a new project:**

1. Select **File** > **New Project...** The New Project Wizard appears.

2. Type **tutorial** in the Project Name field.

3. Enter or browse to a location (directory path) for the new project. A tutorial subdirectory is created automatically.

4. Verify that **HDL** is selected from the Top-Level Source Type list.




5. Click **Next** to move to the device properties page.

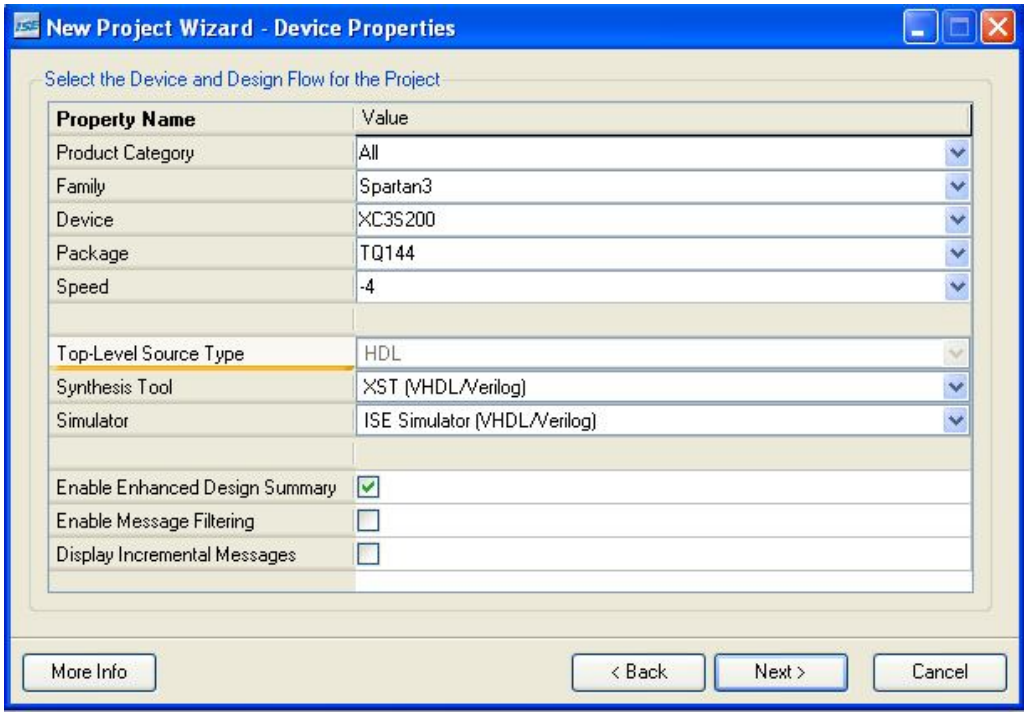
6. Fill in the properties in the table as shown below:

- ✓ Product Category: **All**
- ✓ Family: **Spartan3**
- ✓ Device: **XC3S200**

- ✓ Package: **TQ144** selected.
- ✓ Speed Grade: **-4**
- ✓ Top-Level Module Type: **HDL**
- ✓ Synthesis Tool: **XST (VHDL/Verilog)**
- ✓ Simulator: **ISE Simulator (VHDL/Verilog)**
- ✓ Verify that **Enable Enhanced Design Summary** is

Leave the default values in the remaining fields.

When the table is complete, your project properties will look like the following:




7. Click **Next** to proceed to the Create New Source window in the New Project Wizard.

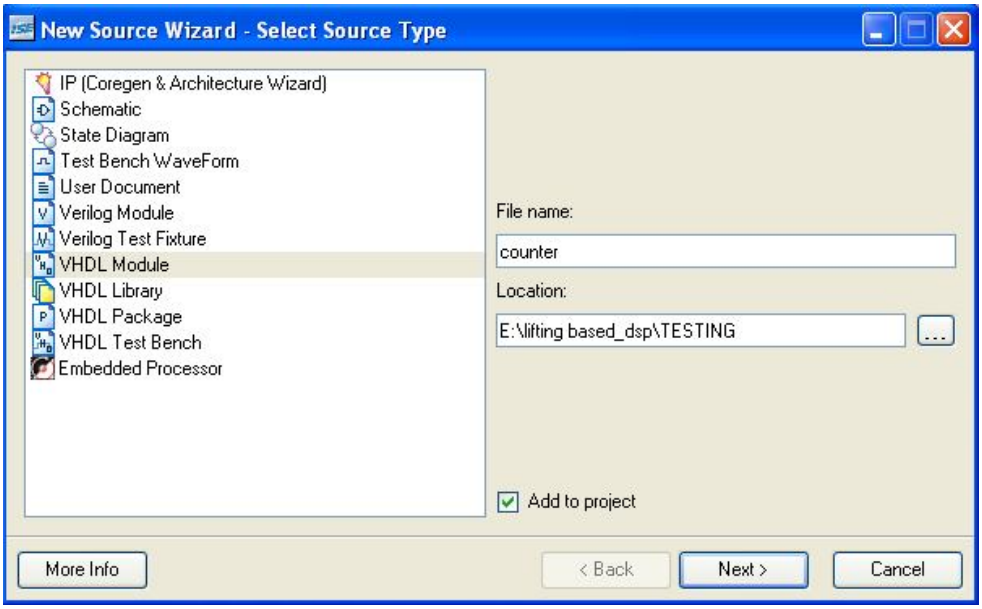
**Create an HDL Source**

You will create the top-level HDL file for your design. Determine the language that you wish to use for the tutorial. Then, continue either to the "Creating a VHDL Source" section below, or creating a "Creating a Verilog Source" similar to VHDL source creating.

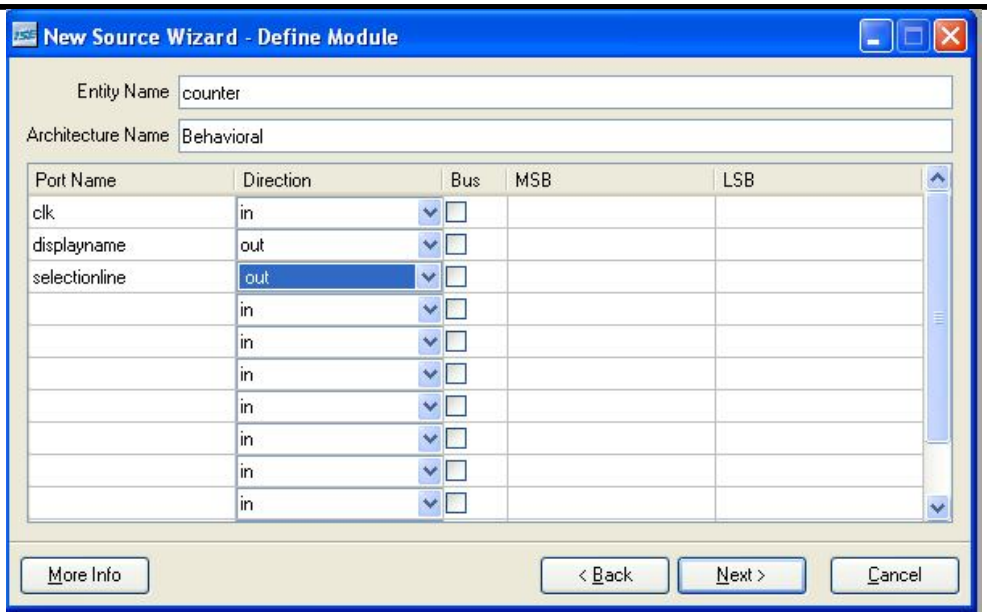
**Creating a VHDL Source**

Create a VHDL source file for the project as follows:

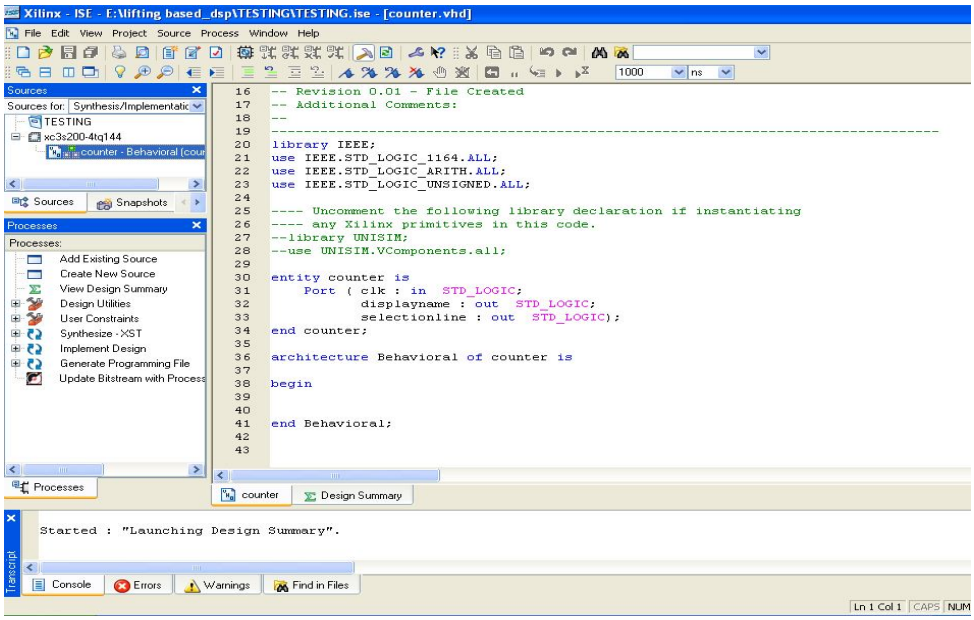
1. Click the **New Source** button in the New Project Wizard.
2. Select **VHDL Module** as the source type.
3. Type in the file name **counter**.
4. Verify that the **Add to project** checkbox is selected.




5. Click **Next**.
6. Declare the ports for the counter design by filling in the port information as shown below:



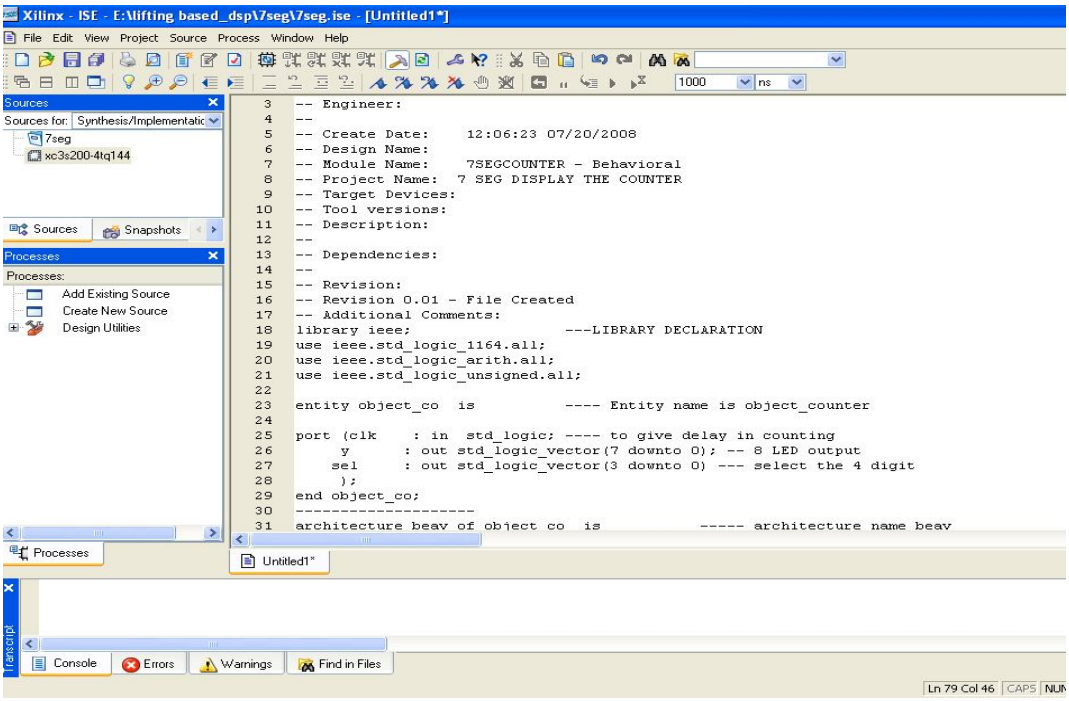

**8.** Click **Next**, then **Finish** in the New Source Information dialog box to complete the new source file template.



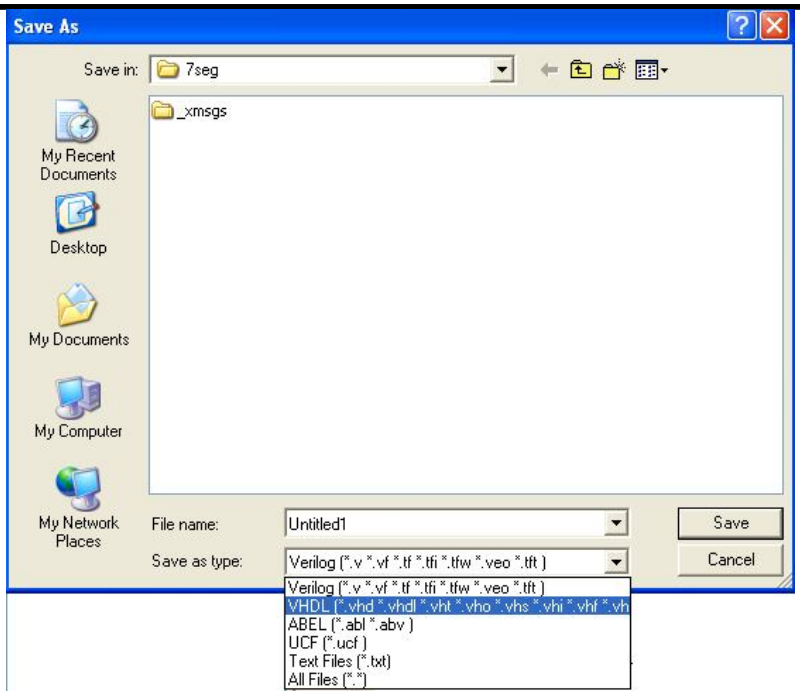

+ Refer Lab Examples

Let us we will see source code for Four Digit 7 segment LED display to configure in Spartan 3 FPGA with synthesis and implementation.

Step 1: copy the above source code and paste it in project window in ISE 8.1i as shown below.

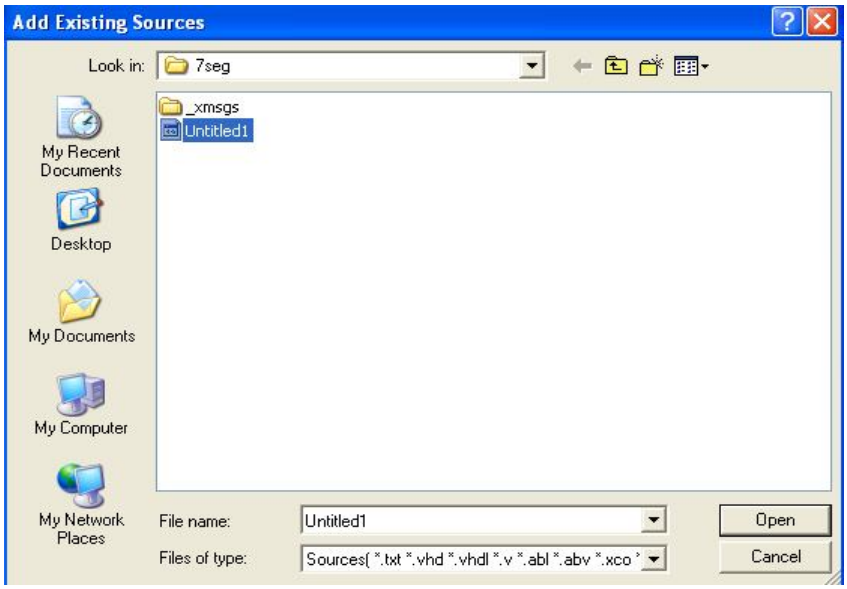



Step 2: Save the file by selecting File → Save the file with .vhd extension.

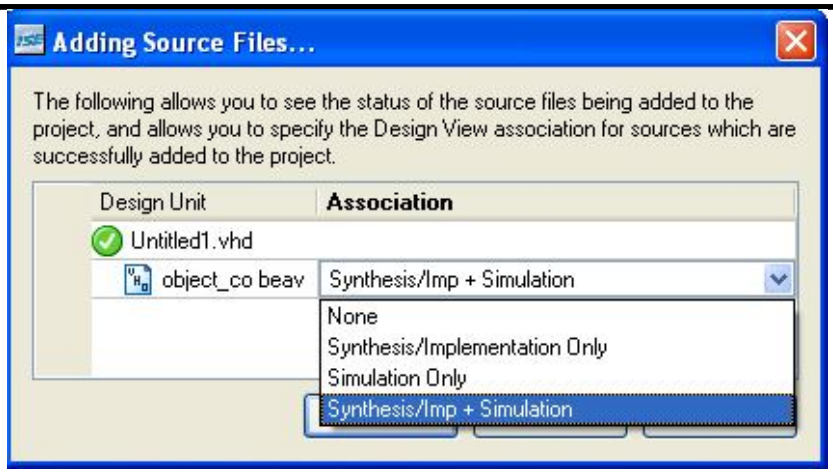



Step 3: Add your source code in your source window place a cursor above the IC number xc3s200-4tq144.

Right click → Add source → Open .vhd file.




Step 4: verify that your code is added with synthesis and implementation.

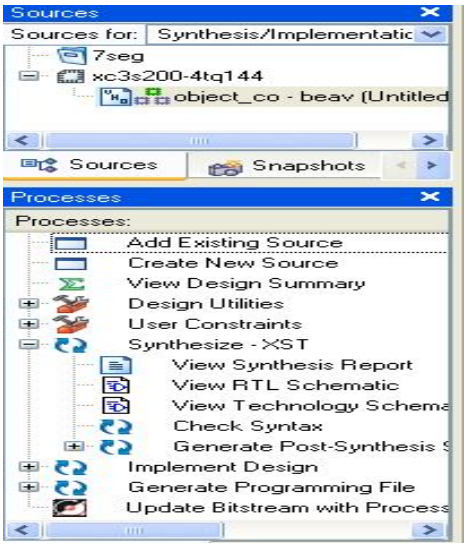



Step 5: Click ok.

Step 6: Checking the Syntax of the 7SEGMENT COUNTER Module

When the source files added complete, check the syntax of the design to find errors and typos.

- Verify that **Synthesis/Implementation** is selected from the drop-down list in the Sources window.
- Select the **object_co** design source in the Sources window to display the related processes in the Processes window.
- Click the "**+**" next to the Synthesize-XST process to expand the process group.




- Double-click the **Check Syntax** process.

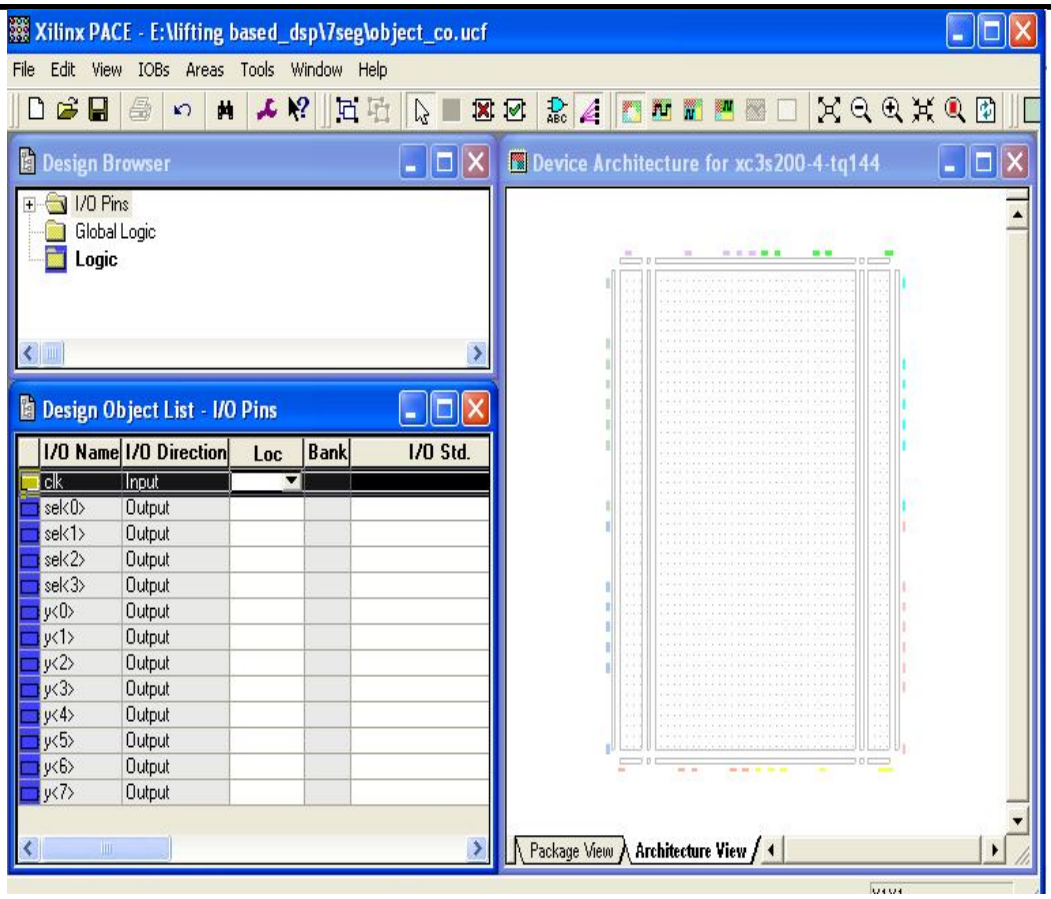
**Note:** You must correct any errors found in your source files. You can check for errors in the Console tab of the Transcript window. If you continue without valid syntax, you will not be able to simulate or synthesize your design.

Step 7: Assigning Pin Location Constraints

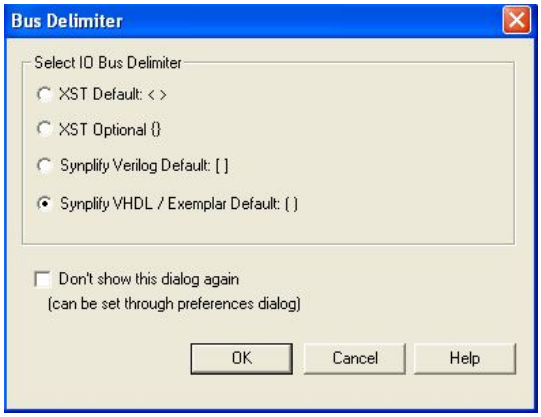
Specify the pin locations for the ports of the design so that they are connected correctly on the Spartan-3 Advance development board.

To constrain the design ports to package pins, do the following before that refer the chapter 4 for giving the pin location.

- Verify that **object_co** is selected in the Sources window.
- Double-click the **Assign Package Pins** process found in the User Constraints process group. The Xilinx Pin out and Area Constraints Editor (PACE) opens.
- Select the **Package View** tab.
- In the Design Object List window, enter a pin location for each pin in the **Loc** column using the chapter 4 refer page number 9.




- Select **File** → **Save**. You are prompted to select the bus delimiter type window will open. Select synplify VHDL/exemplar default [] and click **OK**.
- Close PACE.




Step 8: Implementing the Design

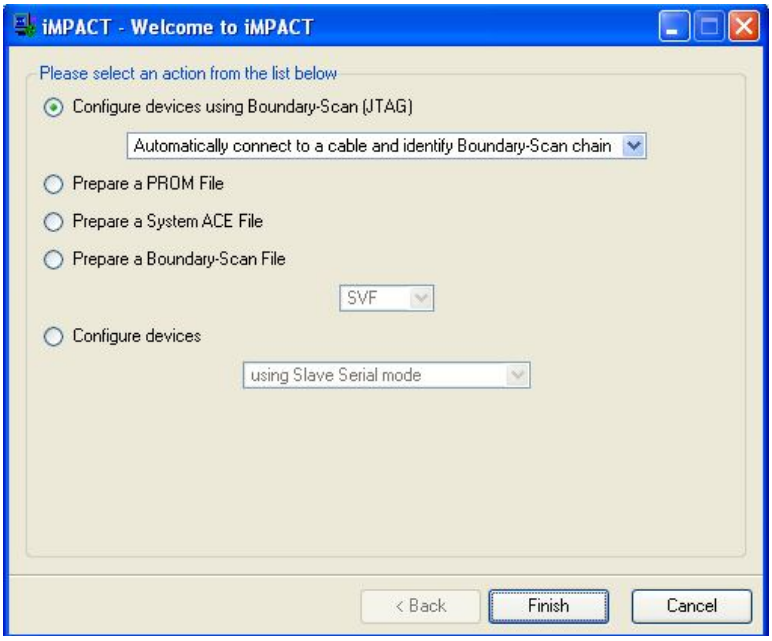
- Select the **object_beh** source file in the Sources window.

- Double-click the **Implement Design** process in the Processes tab.
- Notice that after Implementation is complete, the Implementation processes have a green check mark next to them indicating that they completed successfully without Errors or Warnings.

Step 9: Download Design to the Spartan™-3 Advance Development Board.

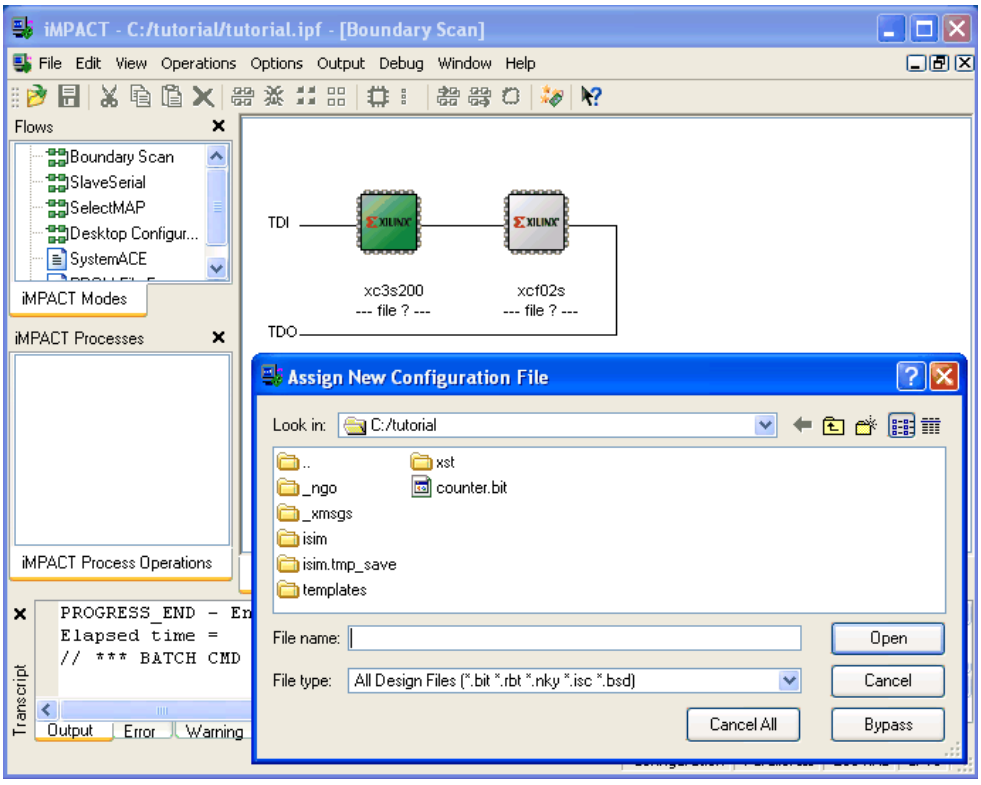
This is the last step in the design verification process.

- Connect the 9V AC/DC power cable to the power input on the advance development board (J1).
- Connect the download cable between the PC and development board (J4).
- Select **Synthesis/Implementation** from the drop-down list in the Sources window.
- Select **object_beh** in the Sources window.
- In the Processes window, click the "**+**" sign to expand the **Generate Programming File** processes.
- Double-click the **Configure Device (iMPACT)** process.




- In the Welcome dialog box, select **Configure devices using Boundary-Scan (JTAG)**.
- Verify that **Automatically connect to a cable and identify Boundary-Scan chain** is selected.
- Click **Finish**.
- If you get a message saying that there are two devices found, click **OK** to continue.
- The devices connected to the JTAG chain on the board will be detected and displayed in the iMPACT window

- The **Assign New Configuration File** dialog box appears. To assign a configuration file to the xc3s200 device in the JTAG chain, select the object_beh.bit file and click **Open.**




- If you get a Warning message, click **OK**.
- Select **Bypass** to skip any remaining devices.
- Right-click on the xc3s200 device image, and select **Program...** The **Programming Properties** dialog box opens.
- Click **OK** to program the device.

When programming is complete, the Program Succeeded message is displayed.


On the board, 4 Digits 7 SEG indicating that the counter is running.

- Close iMPACT without saving.